

IUP Génie Mathématique et Informatique



Langage C : énoncé et corrigé des exercices

Maude Manouvrier

Table des matières

1 ENONCE DES EXERCICES

Les exercices 1 à 9, 16, 22 à 25, 29, 33, et 42 à 43 sont tirés des énoncés de M. B. Quément, Maître de Conférence à l'Université Paris Dauphine. Les exercices 13 à 15, 20 à 21, 30 à 32 sont issus des exercices de Bernard Cassagne - Laboratoire de Génie Informatique - IMAG - Grenoble. Les exercices 17 à 19, 26 à 28, 34 à 41, et à partir de 44, sont tirés des énoncés de R. Cori et G. Bianchi - Module I6 de Bordeaux I - DEUG MIAS.

Les exercices 1 à 16, 20 à 25, 29 à 33, 42 à 43 sont corrigés. Les solutions sont données à la fin du polycopié (voir table des matières).

1.1 EXERCICES FACILES

Exercice 1 Ecrire un programme qui saisit deux entiers et affiche leur produit. Modifier ensuite le programme afin de saisir deux réels.

Exercice 2 Ecrire un programme qui échange deux entiers saisis. Afficher les entiers avant et après l'échange.

Exercice 3 Ecrire un programme qui affiche les code ASCII des lettres et des chiffres sous la forme suivante :

```
caractère = A     code = 65    code hexa = 41
caractère = B     code = 66    code hexa = 42
...
caractère = 1     code = 49    code hexa = 31
...
caractère = 9     code = 57    code hexa = 39
```

Exercice 4 Ecrire un programme qui détermine si un entier saisi est pair ou impair.

Exercice 5 Ecrire un programme qui affiche le plus grand de trois entiers saisis.

Exercice 6 Ecrire un programme qui affiche le plus grand et le plus petit d'une suite d'entiers saisis. Les nombres saisis ne se sont pas conservés en mémoire. La suite se termine avec la valeur 0.

Exercice 7 Ecrire un programme qui détermine tous les diviseurs d'un nombre entier saisi, plus grand que 1.

Exercice 8 Ecrire un programme qui simule l'opération de division entière entre deux entiers positifs a et b. Les deux nombres a et b sont saisis au clavier. On divise le plus grand par le plus petit, sans utiliser l'opérateur /. Afficher le quotient et le reste.

Exercice 9 Ecrire un programme qui multiplie deux entiers positifs a et b selon le principe récursif suivant :

```
a * b = a * (b-1) + a    si b est impair
a * b = (2 * a) * (b/2)  si b est pair et différent de 0

EXEMPLE : 36 * 7 = 36 * 6 + 36
                 = 72 * 3 + 36
                 = 72 * 2 + 108
                 = 144 * 1 + 108
                 = 144 * 0 + 252
                 = 252
```

Ecrire un programme qui lit deux entiers a et b à partir du clavier, et affiche leur produit selon l'algorithme itératif défini ci-dessus. Fournir les résultat tels qu'ils figurent dans l'exemple.

Exercice 10 Ecrire un programme se comportant comme une calculatrice, c'est-à-dire exécutant la boucle sur :

1. Lecture d'une ligne supposée contenant un entier, un opérateur et un entier (ex : 1 + 3). Les opérateurs sont +, -, *, \ et %.
2. Calcul de la valeur de l'expression.
3. Impression du résultat à l'écran.

Exercice 11 Ecrire un programme dans lequel vous :

1. Déclarerez un entier i et un pointeur vers un entier, p,
2. Initialiserez l'entier à une valeur arbitraire et ferez pointer le pointeur sur i,
3. Imprimerez la valeur de i,
4. Modifierez l'entier pointé par p (en utilisant p et non pas i),
5. Et imprimerez la valeur de i une dernière fois.

Exercice 12 Ecrire un programme dans lequel vous déclarerez et initialiserez un tableau d'entiers t avec des valeurs, dont certaines seront nulles. Le programme doit parcourir le tableau et imprimer les index des éléments nuls du tableau (sans utiliser aucune variable de type entier).

Exercice 13 Déclarer et initialiser une matrice [5,5] d'entiers ($iMat$). Ecrire une fonction $affiche_matrice$ qui admette en paramètre une matrice [5,5] et qui imprime ses éléments sous forme de tableau. La procédure $main$ fera un appel à $affiche_matrice$ pour la matrice $iMat$.

Exercice 14 Déclarer un tableau iNb_jours qui doit être initialisé de façon à ce que $iNb_jours[i]$ soit égal au nombre de jours du ième mois de l'année, pour i allant de 1 à 12 ($iNb_jours[0]$ sera inutilisé).

Ecrire une procédure d'initialisation de iNb_jours qui utilisera l'algorithme suivant :

- si i vaut 2, le nombre de jours est de 28,
- sinon, si i est pair et $i <= 7$ ou i impair et $i > 7$ le nombre de jours est de 30,
- sinon le nombre de jours est de 31.

Ecrire une procédure d'impression des 12 valeurs utiles de iNb_jours. La procédure $main$ se contentera d'appeler les procédures d'initialisation et d'impression.
Une variante de ce programme est de réaliser une procédure d'impression pour un mois i donné, et de faire saisir ce mois par l'utilisateur dans le programme principal.

Exercice 15 Ecrire un programme comportant :

1. La déclaration de trois variables globales entières $iHeures$,$iMinutes$,$iSecondes$.
2. Un procédure $affiche_heure$ qui imprimera le message suivant :
   `Il est ... heure(s) ... minute(s) ... seconde(s)`
   En respectant l'orthographe du singulier et du pluriel.
3. Une procédure $saisir_heure$ qui admettra trois paramètres entiers iH,iM et iS, dont elle affectera les valeurs respectivement à $iHeures$,$iMinutes$ et $iSecondes$.
4. Un procédure $tick$ qui incrémentera l'heure d'une seconde.
5. La procédure $main$ sera un jeu d'essais des procédures précédentes.

Exercice 16 Ecrire un programme de saisie d'un tableau d'entiers, d'affichage du tableau et de tri du tableau. La dimension du tableau peut être fixée dans un premier temps, puis demandé à l'usager lors de l'exécution. L'algorithme de tri demandé est le suivant :

- Mettre le plus petit entier en première position du tableau. On opère par comparaison deux à deux entre le premier élément du tableau et un élément d'indice i du tableau. On échange les deux éléments lorsque ceux-ci ne sont pas dans un ordre correct.

– Opérer de la même façon avec le deuxième élément du tableau, puis successivement avec chaque élément du tableau. La dernière comparaison se fait alors entre l'avant-dernier élément et le dernier élément du tableau.

1.2 CHAINES DE CARACTERES

Exercice 17 On utilisera le type : *typedef char mot[NMAX]* où *NMAX* est une constante entière préalablement définie.

Pour cette exercice, vous n'utiliserez aucune des fonctions `strcpy`, `strcmp`, `strncmp`*. Vous pourrez néanmoins utiliser* `strlen`*.*

- Ecrire et tester une fonction *miroir(mot dest,mot src)* rangeant dans le mot *dest* le miroir (mot à l'envers) du mot *src*.
- Ecrire une fonction *void majuscules(mot dest, mot src)* convertissant les lettres minuscules du mot *src* en majuscules et en sauvegardant le résultat dans le mot *dest* (les éventuels caractères autres que les minuscules ne sont pas modifiés).
- Ecrire et tester :
  - une fonction *int est_facteur_gauche(mot facteur_g, mot src)* qui vérifie si *facteur_g* est un facteur gauche de *src*.
    Par exemple "poly" est un facteur gauche de "polycopié"

  - une fonction *int est_facteur(mot facteur, mot src)* qui vérifie si le mot *facteur* est un facteur de *src*.
    Par exemple "poly", "copi", "polycop" sont des facteurs de "polycopié"

  - une fonction *int est_sous_mot(mot sous_mot, mot src)* qui fait la même chose que *est_facteur*, mais de manière récursive.
    Cette fonction donne le même résultat que la fonction *est_facteur*, mais est récursive.

Exercice 18 On utilisera les types suivants :

```
typedef char* mot; /* adresse d'une chaîne de caractère de longueur maximale */
                   /* MOTMAX = 50                                            */

typedef mot *langage; /* tableau de mots contenant au plus LANGMAX=100       */
                      /* éléments dont le dernier a pour valeur NULL         */
```

- Ecrire une fonction *void afficher(langage l)* affichant à l'écran les mots du langage *l*.
- Ecrire une fonction *int appartient(mot m, langage l)* retournant 1 ou 0 suivant que *m* est ou non un mot du langage *l*.
- Ecrire une fonction *int disjoints(langage l1,langage l2)* retournant 1 ou 0 suivant que *l*1 ou *l*2 sont ou non disjoints.
- Ecrire une fonction *int ajouter(mot m, langage l)* qui ajoute le mot *m* au langage *l* et retourne son indice dans le tableau (ou -1 si m=NULL).
- Ecrire une fonction *void reunir(langage l1, langage l2)* qui ajoute au langage *l*2 les mots de *l*1 qui ne s'y trouvaient pas.
- Ecrire une fonction *void enlever(mot m, langage l)* qui enlève le mot *m* au langage *l*.

Exercice 19 Soit le programme suivant :

```
#include <stdio.h>

void main(int argc,char* argv[])
{
  if(argc<3)
  {
     printf("%s : Usage %s <mot-1> <mot-2>\n", argv[0], argv[0]);
     return;
  }
  printf("\t%s\n\t%s\n",argv[2],argv[1]);
}
```

Commenter ce programme. Imaginer ensuite que ce programme soit compilé en un programme exécutable *prog*.

Déterminer l'effet des commandes : *prog*, *prog fleur*, *prog fleur bouillie*, *prog fleur bouillie tourbillon.*

Les exercices qui suivent sont corrigés.

Exercice 20 Déclarer et initialiser deux tableaux de caractères (*cTab*1 et *cTab*2). Puis :

1. Ecrire une fonction *longueur_chaine*1 qui admette en paramètre un tableau de caractères se terminant par un *null*, et qui rende le nombre de caractères du tableau (*null* exclu).
2. Ecrire une fonction *longueur_chaine*2 qui implante la même interface que *longueur_chaine*1, mais en donnant à son paramètre le type de pointeur vers un *char*.
3. La procédure *main* imprimera le nombre d'éléments de *cTab*1 et *cTab*2 par un appel de *longueur_chaine*1 et *longueur_chaine*2.

Exercice 21 Algorithme de codage : On choisit un décalage (par exemple 5), et un a sera remplacé par un f, un b par un g, un c par un h, etc.. On ne cryptera que les lettres majuscules et minuscules sans toucher ni à la ponctuation, ni à la mise en page (caractères blancs, etc.). On supposera que les codes des lettres se suivent de a à z et de A à Z.

1. Déclarer un tableau de caractères *cMessage* initialisé avec le message en clair.
2. Ecrire une procédure *crypt* de cryptage d'un caractère qui sera passé par adresse.
3. Ecrire le programme principal *main* qui activera *crypt* sur l'ensemble du message et imprimera le résultat.

Exercice 22 Ecrire un programme qui calcule le nombre de chiffres, de caractères d'espacement (espace, tabulation, fin de ligne) et tout autres caractère qu'il reçoit en entrée. Exemple :

```
chene% prog < test.c
Chiffres : 93000000010 Espacements : 276 Autres : 785
```

Exercice 23 Lire à partir du clavier une suite de noms ne comportant pas de caractères blancs et afficher le nombre de noms ayant plus de dix caractères. La suite se termine lorsque l'on frappe le mot "fin". Le programme doit comporter une fonction de lecture d'un nom ainsi qu'une fonction de comparaison déterminant si un nom comporte plus de dix caractères.
Vous définirez un tableau de 20 caractères dans le programme principal destiné à recevoir le nom lu au clavier. On supposera que chaque nom ne dépasse pas 20 caractères (vous pouvez inclure un test de vérification).

Vous utiliserez les fonctions de traitement des chaînes de caractères dont le rappel de leur définition est donné ci-après :

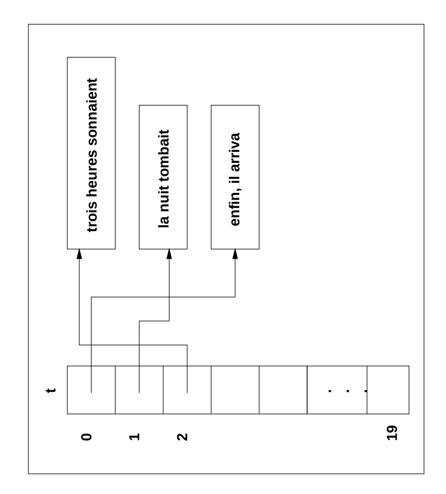


FIG. *1* – **Figure de l'exercice 25.**

```
/* Fonction qui retourne le nombre de caractères de la chaîne */
int strlen (char*)

/* Fonction qui compare deux chaînes s et t.                          */
/* La valeur retour est positive si s est alphabétiquement supérieure à t, */
/* négative si s est alphabétiquement inférieure à t,                 */
/* et 0 si les deux chaînes s et t sont égales.                       */
int strcmp(char* s,char* t)
```

Exercice 24 Ecrire un programme permettant d'effectuer un ensemble d'opérations sur une chaîne de caractères quelconque saisie à partir du clavier. Ce programme est constitué d'un menu comportant le choix de l'opération à effectuer. les opérations sur cette chaîne sont les suivantes :

- fonction *saisir* : elle lit une chaîne de caractères à partir du clavier, elle retourne cette chaîne.
- fonction *afficher* : elle affiche la chaîne argument.
- fonction *inverse* : elle inverse la chaîne argument (elle la modifie donc).
- fonction *mots* : elle compte le nombre de mots de la chaîne. On considère le caractère ' ' (blanc) comme le caractère séparateur de mots. Il peut y avoir plusieurs blancs consécutifs dans la chaîne.

Après chaque opération, le retour au menu s'effectue après l'affichage du message "Frappez une touche pour revenir au menu".

Exercice 25 Saisir à partir du clavier un ensemble de chaînes de caractères dans un tableau de pointeurs. L'ensemble se termine avec la lecture de la chaîne "fin" (qui n'est pas insérée dans le tableau). Le tableau doit rester constamment trié en ordre croissant (ordre alphabétique).

Ainsi avec le tableau de la figure 1 (p. 113), la lecture de la chaîne "enfin, il arriva" entraîne le décalage des pointeurs du tableau à partir du début puisque la chaîne est alphabétiquement inférieure à "la nuit tombait". Notez que les chaînes elles-mêmes ne se déplacent pas.

La mémoire nécessaire pour implanter chaque chaîne doit être allouée au fur et à mesure de la lecture de celles-ci avec la fonction *malloc*. La taille du tableau de pointeurs est fixe et est égale à 20.

Le programme principal sera le suivant :

```
void main()
{ /* tableau de pointeurs */
  char* cTab[20];
  /* Nombre de chaîne actuellement dans le tableau */
  int iNbl;

  /* lecture lit les chaînes et les introduit dans le tableau à leur place */
  iNbl = lecture(cTab,20);

 /* écriture visualise à l'écran le tableau */
  ecriture (cTAb,iNbl);
}
```

Par la suite, modifier la lecture de telle manière qu'après chaque ajout d'une chaîne, le tableau soit entièrement visualisé.

1.3 GESTION DE FICHIERS

Dans les exercices qui suivent, on utilisera la fonction de la bibliothèque standard : *int feof(FILE* f)* **qui retourne EOF ou 0 suivant que le fichier** *f* **(supposé ouvert en lecture) est ou non en dernière position.**

Exercice 26 Ecrire une fonction *int concat(char* sNomFichier, char* sFichier1, char* sFichier2)* concaténant les fichiers de nom $sFichier1$ et $sFichier2$ en un fichier de nom $sNomFichier$, et retournant 0 si tout s'est bien passé, 1 en cas d'erreur (avec si possible un message expliquant l'erreur).

Exercice 27 Ecrire une fonction *int coder(int iDeplacement, char* sNomFichier1, char* sNomFichier2)* qui copie le fichier de nom $sNomFichier1$ dans le fichier de nom $sNomFichier2$ en remplaçant chaque caractère *c* de $sNomFichier1$ par le caractère de valeur *(c+iDeplacement) modulo 256*. Ecrire aussi un programme de test de cette fonction.

Exercice 28 Ecrire un programme *void stat_freq(char* sFichier)* affichant tous les caractères figurant dans le fichier texte $sFichier$, rangés par fréquences décroissantes, et la fréquence de chacun de ces caractères. Vous pouvez découper ce programme en plusieurs sous-fonctions.

Les exercices qui suivent sont corrigés.

Exercice 29 Soit un fichier de données structuré en une suite de lignes contenant chacune un nom de personne, un nom de pièce, un nombre et un prix. Exemple :

```
Dupons Villebrequin 10 1000
Durant Brosse       20 567
```

Ecrire une procédure *main* dans laquelle on déclarera les variables suivantes :

- $cNom$ et $cArticle$: tableaux de 80 caractères,
- $iNombre$ et $iPrix$: entiers.

Le corps de la procédure consistera en une boucle dont chaque itération lira une ligne et l'imprimera. La lecture se fera par un appel à $fscanf$ affectant les 4 champs de la ligne aux 4 variables $cNom$, $cArticle$, $iNombre$ et $iPrix$. L'écriture consistera à imprimer $cNom$, $cArticle$ et le produit $iNombre * iPrix$.

Exercice 30 Soit un fichier de données identiques à celui de l'exercice précédent. Ecrire une procédure *main* qui :

1. Lise le fichier en mémorisant son contenu dans un tableau de structures, chaque structure permettant de mémoriser le contenu d'une ligne (nom, article, nombre et prix).
2. Parcoure ensuite ce tableau en imprimant le contenu de chaque structure.

Exercice 31 Faire le même exercice que le précédent, mais en mémorisant le fichier de données, non pas dans un tableau de structures, mais dans une liste de structures chaînées.

Exercice 32 Modifier le programme précédent tel que :

1. En écrivant la procédure d'impression d'une structure *commande*, procédure qui admettra en paramètre un pointeur vers une telle structure.
2. En écrivant une fonction de recherche de commande maximum (celle pour laquelle le produit nombre*prix est maximum). Cette fonction admettra en paramètre un pointeur vers la structure *commande* qui est en tête de la liste complète, et rendra un pointeur vers la structure recherchée.
3. Le programme principal sera modifié de manière à faire appel à la fonction de recherche de commande maximum et à imprimer cette commande.

1.4 LISTE CHAINEE

Le premier exercice est corrigé.

Exercice 33 Ecrire un programme qui gère les listes chaînées. Pour cela, vous créerez un type de structure de liste chaînée dont les éléments sont des entiers. Vous créerez également un type de structure contenant trois pointeurs, *prem*, *der* et *cour* permettant d'accéder respectivement au premier et dernier élément, ainsi qu'à l'élément courant (cf. poly).

Le programme se composera de plusieurs fonctions :

- Une fonction de création *creer_liste* qui alloue la mémoire nécessaire à une liste et à la structure des trois pointeurs associés.
- Une fonction *insrer_apres* qui prend en paramètre une liste et un entier. Cette fonction insère l'entier dans la liste, après l'élément courant (pointé par *cour*).

Vous testerez votre programme sous UNIX, et fournirez un exemple ou deux d'exécution.

Dans les exercices qui suivent, on utilisera les types de données suivants :

```
typdef struct {
               char sNom[30];
               int iAnnee_naissance;
               char sTelephone[20];
             } personne;
```

```
typedef struct item
         { personne entree;
           struct item *suivant;
         } cellule;
```

```
typedef cellule *liste;
```

Exercice 34 Ecrire une fonction *int saisir(personne *individu)* saisissant au clavier un objet de type *personne*. Cette fonction retournera 0 si l'utilisateur entre un symbole spécial pour signifier qu'il n'a pas de données à saisir, 1 sinon.

Exercice 35 Ecrire une fonction *int est_identique(personne individu1, personne individu2)* retournant 1 si les deux objets *individu*1 et *individu*2 ont les mêmes valeurs.

Exercice 36 Ecrire une fonction *cellule *creer(personne individu)* retournant l'adresse d'un objet de type *cellule* dont les champs *entree.sNom*, *entree.iAnnee_naissance*, *entree.sTelephone* ont les valeurs respectives des champs correspondants de *individu*.

Exercice 37 Ecrire une fonction *void afficher(liste l)* affichant à l'écran une représentation lisible des éléments de la liste *l*.

Exercice 38 Ecrire une fonction *cellule *position(personne individu, liste l)* retournant l'adresse de la première cellule de *l* dont les valeurs des champs de *individu* sont identiques, et NULL si cette cellule n'existe pas.

Exercice 39 Dans cet exercice, on utilisera le type :

```
typedef struct info {
                     int iAge;
                     char* sNom;
                    }

typedef struct cellule
        {  struct info individu;
           struct cellule *suivant;
        } *liste;
```

```
/* Déclaration de variable */
liste prem;
int AGEMAX = 125;
```

Ecrire une fonction *separation(int iAgeMaxJeune)* qui, sans recopier de cellules, éclate la liste *prem* en deux listes : celle des individus d'âge inférieur à la valeur *iAgeMaxJeune*, dont l'adresse sera contenue dans une variable globale *jeunes*, et celle des individus d'âge supérieur ou égal à *iAgeMaxJeune*, dont l'adresse sera contenue dans une variable globale *vieux*.

Exercice 40 Dans cet exercice, on représente une liste d'entiers par une structure de liste chaînée circulaire telle que :

```
typedef struct cellule {
                         int iVal;
                         struct cellule *suivant;
                       } cellule;

typedef cellule *liste;
```

En raison de la circularité de la liste, dans aucune cellule, le champ *suivant* ne vaut *NULL*.

On suppose que la liste est donnée par l'adresse de la première cellule, et que l'ajout d'un nouvel élément se fait en tête de liste.

Ecrire les fonctions suivantes :

- Recherche d'un élément dans une liste circulaire (retournant 1 si l'élément appartient 0 sinon).
- Ajout d'un élément à une liste circulaire.
- Suppression d'un élément dans une liste circulaire.

Exercice 41 On reprend l'exercice précédent, en supposant maintenant que la liste est donnée par l'adresse de sa dernière cellule. L'ajout d'un élément se fait toujours en tête de liste. Reprendre les fonctions précédentes avec cette nouvelle configuration.

1.5 PILE ET FILE

Ces exercices sont corrigés.

Exercice 42 Ecrire un programme qui gère une pile à l'aide d'une liste chaînée. Pour cela, vous créerez un type de structure de pile à l'aide d'une liste chaînée dont le éléments sont des entiers. Le pointeur d'un élément de la pile pointe vers l'élément précédent.

Le programme se composera de plusieurs fonctions :
- Une fonction de création *creer_pile* qui retourne une pointeur de type *pile*, nul.
- Une fonction *vide* qui retourne 0 si la pile, passée en paramètre, est non vide, et un nombre différent de 0 dans le cas contraire.
- Une fonction *sommet* qui retourne le sommet de la pile passée en paramètres.
- Une fonction *empiler* qui empile l'entier $iVal$, passé en paramètre, à la pile p, également passée en paramètre.
- Une fonction *desempiler* qui supprime le sommet de la pile p, passée en paramètre. La mémoire occupée par le précédent sommet de la pile est libérée.
- Une fonction *afficher_recursive* qui affiche le contenu de la pile de manière récursive.

Vous testerez votre programme sous UNIX, et fournirez un exemple ou deux d'exécution.

Exercice 43 Ecrire un programme qui gère une file de caractères en anneau continu (voir poly de cours). Pour cela, vous créerez un type de structure de file contenant quatre pointeurs sur des chaînes de caractères. Ces pointeurs représenteront respectivement le début et la fin de la zone mémoire allouée pour la file, ainsi que la tête et la queue de la file.

Le programme se composera de plusieurs fonctions :
- Une fonction de création *creer_file* qui retourne une file. Cette fonction alloue la zone mémoire nécessaire à la file, ainsi que les pointeurs de la file. A la création les pointeurs de tête et de début pointent au début de la zone mémoire. Les pointeurs de queue et de fin pointent à la fin de la zone. On passera en paramètre de la fonction la taille mémoire (c-à-d le nombre de caractères maximum) de la file.
- Une fonction *plusun* qui avance un pointeur de type $char*$, passé en paramètre, d'un espace mémoire dans la file.
- Une fonction *vide* qui retourne 0 si la file est non vide (un nombre supérieur à 0 sinon). Cette fonction fait appel à la fonction *plusun*. Une file est vide lorsque l'emplacement mémoire qui suit la queue de la file est l'adresse de la tête de file.
- Une fonction *pleine* qui retourne 0 si la file est non pleine (un nombre supérieur à 0 sinon). Cette fonction fait appel à la fonction *plusun*. Une file est pleine lorsque le deuxième emplacement mémoire qui suit la queue de la file est l'adresse de la tête de file. Une file est non pleine lorsqu'il y a au moins un emplacement mémoire vide entre la queue et la tête.
- Une fonction *lire* qui retourne le caractère situé à la tête de la file.
- Une fonction *avancer* qui permet de faire avancer la tête de la file d'un espace mémoire. Elle fait appel à *plusun*. Si la file est vide, la fonction affiche un message d'erreur[1].
- Une fonction *ajouter* qui ajoute un caractère en queue de la file (et déplace donc la queue d'un espace mémoire). Si la file est pleine, la fonction affiche un message d'erreur.
- Une fonction *afficher* qui affiche la file.

Vous testerez votre programme sous UNIX, et fournirez un exemple ou deux d'exécution.

1. Vous pouvez réaliser une fonction *erreur* qui prend en paramètre une chaîne de caractères (le message d'erreur) et l'affiche.

1.6 ARBRES BINAIRES

Dans tous les exercices qui suivent, on utilisera la représentation d'une expression arithmétique par un arbre binaire, au moyen des données suivantes :

```
typedef union {
                int iNombre;
                char cSigne;
              } Type_valeur;
```

```
typedef struct Sommet
          { Type_valeur valeur;
            struct Sommet *fils_g, *fils_d;
          } Sommet;
```

```
typedef Sommet *Arbre;
```

Exercice 44 Ecrire une fonction *Arbre construire_feuille(int iEntier)* retournant un arbre correspondant à l'expression arithmétique constituée d'un seul entier (c'est-à-dire un arbre contenant uniquement une feuille, sommet sans fils gauche ni fils droit).

Exercice 45 Ecrire une fonction *Arbre construire_ noeud(char cOperateur, Arbre gauche, Arbre droit)* retournant un arbre dont la racine est la valeur *sOperateur* et dont les fils droit et gauche sont respectivement *droit* et *gauche*.

Exercice 46 Ecrire les fonctions *void infixe(Arbre tree), void prefixe(Arbre tree)* et *void postfixe(Arbre tree)* en affichant en notation classique (avec parenthèses), en notation polonaise préfixée et en notation polonaise postfixée l'expression arithmétique représentée par *tree*.

```
Par exemple, l'arbre :
      *
     / \
    /   \
   +     -
  / \   / \
 12 15 *   3
      / \
     7  10
```

```
En infixe, s'écrit (12+15)*((7*10)-3)
Le prefixe est *+1215-*7103
Le postfixe doit afficher 1215+710*3-*
```

Exercice 47 Ecrire une fonction *int eval(Arbre tree)* retournant la valeur de l'expression représentée par *tree*.

Exercice 48 Ecrire une fonction *Arbre construire(char* sExpression, int iIndiceDebut, int iIndiceFin)* qui construit l'arbre binaire correspondant à l'expression arithmétique se trouvant entre les indices *iIndiceDebut* et *iIndiceFin* de la chaîne *sExpression*. *sExpression* est écrite sous forme parenthèsée. Vous pouvez décomposer cette fonction en sous fonctions. Ecrire notamment une fonction *int indice_fin(char* sExpression, int iIndice)* qui calcule l'indice où se trouve une parenthèse fermante correspondant à *sExpression[iIndice]='('*.

Dans les exercices qui suivent, on utilisera le type suivant :

```
typedef struct noeud {
                       char* sValeur;
                       struct noeud *fils_g, *fils_d;
                     } * arbre;
```

Exercice 49 Ecrire une fonction *arbre ajouter(char* sAjout, arbre tree)* ajoutant un sommet étiqueté *sAjout* à l'arbre *tree* et retournant l'arbre obtenu. Les chaînes de caractères sont ordonnées dans l'ordre lexicographique dans l'arbre binaire de recherche. Vous pouvez vous aider en écrivant une fonction *arbre feuille(char* sAjout)*.

Exercice 50 Ecrire une fonction *void afficher(arbre tree)* affichant de manière récursive les étiquettes de l'arbre.

Exercice 51 Ecrire une fonction *void main(int argc,char* argv[])* affichant dans l'ordre lexicographique les chaînes du tableau *argv*. Vous testerez ce programme.

Exercice 52 Ecrire une fonction *arbre supprimer(char* sChaine, arbre tree)* retirant le somme étiqueté *sSommet* de *tree* et retournant l'arbre obtenu.

2 CORRECTIONS DES EXERCICES

Une solution est proposée ci-dessous, pour les exercices 1 à 16, 20 à 25, 29 à 33, 42 à 43.

EXERCICE 1

```
/* Inclusion de la bibliothèque standard */
#include <stdio.h>
/*****************************************************************/
/*                    Programme principal                        */
/*              Affichage du produit de deux entiers             */
/*****************************************************************/
void main()
{ /* Déclaration de deux entiers */
  int iFacteur1, iFacteur2;

  /* Saisie au clavier du premier entier */
  printf("valeur du premier facteur :"); scanf("%d",&iFacteur1);

  /* Saisie au clavier du deuxième entier */
  printf("valeur du deuxième facteur :"); scanf("%d",&iFacteur2);

  /* Affichage du produit */
  printf("Produit de %d et %d = %d",iFacteur1,iFacteur2,iFacteur1*iFacteur2);
}
```

Pour afficher le produit de deux entiers, il suffit de modifier le type des facteurs (*float*), ainsi que leur format (%*f*) d'affichage et de lecture (dans le *scanf*).

EXERCICE 2

```
/* Inclusion de la bibliothèque standard */
#include <stdio.h>
/*****************************************************************/
/*                    Programme principal                        */
/*                    Echange de deux entiers                    */
/*****************************************************************/
void main()
{ /* Déclaration de deux entiers */
  int iEntier1, iEntier2;
  /* Déclaration d'un entier pour l'échange */
  int iEchange;

  /* Saisie au clavier du premier entier */
  printf("valeur du premier entier :"); scanf("%d",&iEntier1);
  /* Saisie au clavier du deuxième entier */
  printf("valeur du deuxième entier"); scanf("%d",&iEntier2);

  /* Echange des entiers iEntier1 et iEntier2 */
  iEchange=iEntier1;iEntier1=iEntier2;iEntier2=iEchange;
  /* Affichage des entiers échangés */
  printf("Après échange, le 1er entier = %d, et le 2ème = %d",iEntier1,iEntier2);
}
```

EXERCICE 3

```
#include <stdio.h>

/*****************************************************************/
/*                      Programme principal                      */
/*           Affichage du code ASCII et hexa des caractères      */
/*****************************************************************/
void main()
{ /* Déclaration d'un caractère (compteur)*/
  char cCompteur;

  /* Boucle sur tous les caractères de l'alphabet */
  for(cCompteur='A';cCompteur<='Z';cCompteur++)
    /* Affichage du code ASCII et du code hexadécimal du caractère */
    printf("caractère = %c    code = %d    code hexa = %x\n",cCompteur,cCompteur,cCompteur);

  /* Boucle sur tous les caractères numériques de 1 à 9 */
   for(cCompteur='1';cCompteur<='9';cCompteur++)
    /* Affichage du code ASCII et du code hexadécimal du caractère */
    printf("caractère = %c code = %d code hexa = %x\n",cCompteur,cCompteur,cCompteur);
}
```

EXERCICE 4

```
#include <stdio.h>

/*****************************************************************/
/*                      Programme principal                      */
/*                 Test de parité sur un entier                  */
/*****************************************************************/
void main()
{ /* Déclaration d'un entier */
  int iEntier;

  /* Saisie au clavier de l'entier */
  printf("valeur :"); scanf("%d",&iEntier);

  /* Si le reste de la division de iEntier par 2 est non nul */
  if(iEntier%2)
    printf("entier impair");

  /* Si le reste de la division de iEntier par 2 est nul */
  else
    printf("entier pair");
}
```

EXERCICE 5

```
#include <stdio.h>

/****************************************************************/
/*                      Programme principal                     */
/*             Affichage du max de trois nombres entiers        */
/****************************************************************/
void main()
{
  /* Déclaration de trois entiers */
  int iEntier1, iEntier2, iEntier3;
  /* Déclaration d'un entier, le max des trois */
  int iMax;

  /* Saisie des trois entiers iEntier1,iEntier2 et iEntier3 */
  printf("valeur de iEntier1 :"); scanf("%d",&iEntier1);
  printf("valeur de iEntier2 :"); scanf("%d",&iEntier2);
  printf("valeur de iEntier3 :"); scanf("%d",&iEntier3);

  /* Si iEntier1 est plus grand que iEntier2 */
  if(iEntier1>=iEntier2)
     /* Alors le max de iEntier1 et iEntier2 est iEntier1 */
     iMax = iEntier1;

  /* Sinon (i.e. si iEntier1 est plus petit que iEntier2) */
  else
     /* Alors le max est iEntier2 */
     iMax = iEntier2;

  /* Si iEntier3 est plus grand que le max  */
  if(iEntier3>=iMax)
      /*Alors le max devient iEntier3  */
      iMax = iEntier3;

  /*Affichage du max de iEntier1 iEntier2 iEntier3 */
  printf("Le maximum de %d, %d et %d est : %d",iEntier1,iEntier2,iEntier3,iMax);
}
```

EXERCICE 6

```
#include <stdio.h>

/****************************************************************/
/*                    Programme principal                       */
/*   Affichage du plus grand entier et du plus petit entier     */
/*                 d'une suite d'entiers                        */
/****************************************************************/
void main()
{ /* Déclaration du plus grand et du plus petit entier de la suite */
  int iPlusGrand;
  int iPlusPetit;

  /* Déclaration d'un entier saisi */
  int iSaisi;

  /* Saisie de l'entier iSaisi */
  printf("entier :"); scanf("%d",&iSaisi);

  /* Initialisation de iPlusPetit et iPlusGrand à iSaisi */
  iPlusPetit = iPlusGrand = iSaisi;

  /* Tant que l'utilisateur saisit un entier différent de 0 */
  while(iSaisi)
  {
    /* Si iPlusGrand est plus petit que iSaisi, */
    if(iPlusGrand<iSaisi)
      /* Alors iPlusGrand devient iSaisi       */
      iPlusGrand = iSaisi;

    /* Sinon (iPlusGrand est plus grand que iSaisi)          */
    else
      /* Si iPlusPetit est plus grand que iSaisi,            */
      if(iPlusPetit>iSaisi)
        /* alors iPlusPetit devient iSaisi                   */
        iPlusPetit = iSaisi;

    /* Nouvelle saisie d'un entier */
    printf("entier :"); scanf("%d",&iSaisi);

  } /* Fin du while(iSaisi) */

  /* Affichage de iPlusPetit et iPlusGrand */
  printf("entier le plus grand = %d et entier le plus petit = %d",iPlusGrand,iPlusPetit);
}
```

EXERCICE 7

```
#include <stdio.h>

/***************************************************************/
/*                    Programme principal                      */
/*                    Diviseur d'un entier                     */
/***************************************************************/
void main()
{ /* Déclaration et initialisation du diviseur */
  int iDiviseur=1;

  /* Déclaration d'un entier à saisir*/
  int iSaisi;

  /* Saisie de iSaisi */
  printf("entier :"); scanf("%d",&iSaisi);

  /* Boucler */
  do
   { /* Si le reste de la division entière de iSaisi par iDiviseur est nul */
     if (iSaisi%iDiviseur == 0) /* Equivalent à if(!(iSaisi%iDiviseur)) */
         /*Affichage du diviseur */
         printf("%d est un diviseur de %d\n",iDiviseur,iSaisi);

     /* Sinon, on ne fait rien */

     /* Incrémentation de iDiviseur */
     iDiviseur++;
   }
  /* Tant que iDiviseur est plus petit que iSaisi */
  while(iDiviseur<=iSaisi);
}
```

EXERCICE 8

```
#include <stdio.h>

/****************************************************************/
/*                     Programme principal                      */
/*                  Simulation de la division                   */
/****************************************************************/
void main()
{ /* Déclaration de deux entiers à diviser                      */
  int iSaisi1,iSaisi2;
  /* Déclaration et initialisation du quotient de la division   */
  int iQuotient=0;
  /* Déclaration de deux entiers tampons iPlusGrand et iPlusPetit */
  int iPlusGrand,iPlusPetit;
  /* Déclaration d'une sauvegarde                               */
  int iSauve;

  /* Saisie de iSaisi1 et iSaisi2 */
  printf("Premier entier  :"); scanf("%d",&iSaisi1);
  printf("Deuxième entier :"); scanf("%d",&iSaisi2);

  /* Si Saisi1 est plus grand que iSaisi2 */
  if(iSaisi1>iSaisi2)
    { /* iPlusGrand et iSauve sont égaux à iSaisi1 (le plus grand des 2) */
      /* et iPlusPetit est égal à iSaisi2 (le plus petit des 2)         */
      iPlusGrand=iSaisi1;iPlusPetit=iSaisi2;iSauve=iSaisi1;
    }

  /* Si iSaisi1 est plus petit ou égal à iSaisi2 */
  else
    { /* iPlusGrand et iSauve sont égaux à iSaisi2 (le plus grand des 2) */
      /* et iPlusPetit est égal à iSaisi1 (le plus petit des 2)         */
      iPlusGrand=iSaisi2;iPlusPetit=iSaisi1;iSauve=iSaisi2;
    }

  /* Tant que iPlusGrand est plus grand que iPlusPetit */
  while(iPlusGrand>=iPlusPetit)
    { /* Incrémentation de iQuotient */
      iQuotient++;

      /* Soustraction de iPlusPetit à iPlusGrand */
      iPlusGrand-=iPlusPetit;

    }/* Fin du while(iPlusGrand>iPlusPetit) */

   /* Affichage du résultat */
   printf("quotient de %d par %d = %d\n",iSauve,iPlusPetit,iQuotient);
   printf(" reste = %d",iPlusGrand);
}
```

EXERCICE 9

```
#include <stdio.h>
/*****************************************************************/
/*                     Programme principal                       */
/*                 Multiplication égyptienne                     */
/*****************************************************************/
void main()
{ /* Déclaration des entiers à multiplier */
  int iEntier1,iEntier2;
  /* Déclaration des entiers tampons le plus grand et le plus petit */
  int iPlusGrand,iPlusPetit;
  /* Déclaration et initialisation du résultat de la multiplication */
  int iResultat=0;

  /* Saisie de iEntier1 et iEntier2 */
  printf("iEntier1 :"); scanf("%d",&iEntier1);
  printf("iEntier2 :"); scanf("%d",&iEntier2);

  /* Si iEntier1 est plus grand que iEntier2 */
  if(iEntier1>iEntier2)
    { /* Initialisation de iPlusGrand et iPlusPetit */
      iPlusGrand=iEntier1;iPlusPetit=iEntier2;
    }

  /* Si iEntier1 est plus petit ou égale à iEntier2 */
  else
    { /* Initialisation de iPlusGrand et iPlusPetit */
      iPlusGrand=iEntier2;iPlusPetit=iEntier1;
    }

  /* Affichage de "iPlusGrand * iPlusPetit" */
  printf("%d * %d ",iPlusGrand,iPlusPetit);

  /* Tant que iPlusPetit est non nul */
  while(iPlusPetit)
  { /* Si iPlusPetit est impair */
    if(iPlusPetit%2)
    { /* On décrémente iPlusPetit */
      iPlusPetit--;
      /* On ajout iPlusGrand au résultat */
      iResultat+=iPlusGrand;
      /* Affichage du résultat intermédiaire */
      printf("= %d * %d + %d\n",iPlusGrand,iPlusPetit,iResultat);
    } /* Fin du if((iPlusPetit/2)*2 != iPlusPetit)*/

   /* iPlusPetit est impair ou on l'a décrémenté de 1 lorsqu'il était pair */
   /* Multiplication de iPlusGrand par 2 */
   iPlusGrand*=2;
   /*Division de iPlusPetit par 2 */
   iPlusPetit/=2;
```

```
    /* Affichage du résultat intermédiaire */
    if(iPlusPetit) printf("= %d * %d + %d\n",iPlusGrand,iPlusPetit,iResultat);
  } /* Fin du while(iPlusPetit)

  /* Affichage du résultat */
  printf("= %d\n",iResultat);
}
```

Le nombre d'itérations dépend de la valeur de *iPlusPetit*. Et, afin d'améliorer l'algorithme, il faut prendre, pour *iPlusPetit*, le minimum des deux nombres i*Entier*1 et *iEntier*2 entrés au clavier. De plus, quand *iPlusPetit* est impair, il est pair à l'itération suivante.
Il est donc inutile de tester la parité de *iPlusPetit* à chaque fois qu'il est impair. On peut systématiquement effectuer le calcul dans l'hypothèse pair.

EXERCICE 10

```
#include <stdio.h>

/* Déclaration de constante Booléenne */
const int VRAI = 1;
const int  FAUX  = 0;

/*******************************************************************/
/*                        Programme principal                      */
/*                           Calculatrice                          */
/*******************************************************************/
void main()
{ /* Déclaration des opérandes */
  int iEntier1,iEntier2;
  int iResultat;
  /* Déclaration de l'opérateur */
  char cOperateur;
  /* Déclaration d'un booléen d'impression du résultat */
  int iAutorisationImp;

  /* Boucle infinie */
  while(1)
  { /* Saisie de l'opération */
    scanf("%d %c %d",&iEntier1,&cOperateur,&iEntier2);

    /* Initialisation de iAutorisationImp */
    iAutorisationImp = VRAI;
```

```
    /* Test sur l'opérateur */
    switch(cOperateur)
    { case '+': iResultat = iEntier1 + iEntier2;break;
      case '-': iResultat = iEntier1 - iEntier2;break;
      case '*': iResultat = iEntier1 * iEntier2;break;

      case '/':
                /* Si iEntier2 est nul */
                if (iEntier2 == 0)
                { /* Impression d'un message d'erreur */
                  printf("Division par zéro");
                  /* Affectation de FAUX à iAutorisationImp */
                  iAutorisationImp = FAUX;
                }
                /* Si iEntier2 est non nul */
                else iResultat = iEntier1 / iEntier2;
                break;

      case '%': iResultat = iEntier1 % iEntier2; break;
      default : { printf("L'opérateur %c est incorrect",cOperateur);
                  iAutorisationImp=FAUX;
                }
    } /* Fin du switch */

  /* Si on peut imprimer le résultat, on l'imprime */
  if(iAutorisationImp) printf("résultat :%d\n",iResultat);
  }/* Fin du while(1) */
}
```

EXERCICE 11

```
#include <stdio.h>
/*****************************************************************/
/*                      Programme principal                      */
/*                   Affichage contenu pointeur                  */
/*****************************************************************/
void main()
{ /* Déclaration d'un entier pointé */
  int iEntier;
  /* Déclaration d'un pointeur sur un entier */
  int* iPointeur = NULL;

  /* Initialisation de iEntier */
  scanf("%d",&iEntier);

  /* iPointeur pointe sur iEntier */
  iPointeur = &iEntier;

  /* Affichage de la valeur de iEntier */
  printf("valeur de iEntier avant : %d\n",iEntier);
```

```
  /* Changement de la valeur du contenu de iPointeur */
  (*iPointeur)+=2;

  /* Affichage de la valeur de iEntier */
  printf("valeur de iEntier après : %d\n",iEntier);
}
```

EXERCICE 12

```
#include <stdio.h>

/* Déclaration de constantes */
/* Taille maximum du tableau */
const int  TAILLE_MAX  = 10;

/* Déclaration de variables globales */
/* Tableau statique d'entiers avec éléments nuls */
int iTableau[TAILLE_MAX]={3,6,0,8,97,0,5,6,0,8};

/*****************************************************************/
/*                       Programme principal                     */
/*  Trouver les index des éléments nuls d'un tableau d'entiers   */
/*****************************************************************/
void main()
{ /* Déclaration de pointeurs sur des entiers */
  int* iPointeurDeb = NULL;
  int* iPointeurFin = NULL;
  int* iCompteur = NULL;

  /* iPointeurDeb pointe sur le début du tableau */
  iPointeurDeb = &iTableau[0];
  /* iPointeurFin pointe sur le dernier élément du tableau */
  iPointeurFin = &iTableau[TAILLE_MAX -1];

  /* Pour tous les éléments du tableau, on boucle sur les adresses */
  for(iCompteur=iPointeurDeb;iCompteur<=iPointeurFin;iCompteur++)
    /* Si le contenu de iCompteur est nul on affiche l'index */
    if(*iCompteur==0) printf("%d\n",iCompteur-iPointeurDeb);
  printf("\n");
}
```

EXERCICE 13

```
#include<stdio.h>

/* Définition d'une constante désignant le nombre de lignes de la matrice */
const int NB_LIGNES = 5;

/* Définition d'une variable globale matrice [5,5]                      */
int iMat[NB_LIGNES][NB_LIGNES] =
    {
    {0,1,2,3,4},
    {10,11,12,13,14},
    {20,21,22,23,24},
    {30,31,32,33,34},
    {40,41,42,43,44}
    };

/* Fonction qui imprime une matrice NB_LIGNES * NB_LIGNES, NB_LIGNES constante */
/* Paramètres d'entrée : un tableau NB_LIGNES * NB_LIGNES                     */
/* Paramètres de sortie : rien                                                */
void affiche_matrice(int iTab[NB_LIGNES][NB_LIGNES])
{ /* Déclaration de deux compteurs */
  int iLigne, iColonne;

  /* Pour toutes les lignes de la matrices */
  for(iLigne=0;iLigne<NB_LIGNES;iLigne++)
  { /* Pour toutes les colonnes de la matrice */
    for(iColonne=0;iColonne<NB_LIGNES;iColonne++)
      /* Affichage de l'élément de la matrice situé à la colonne iColonne */
      /* et à la ligne iLigne                                             */
      printf("%d ",iTab[iLigne][iColonne]);
    printf("\n");
  }
}

/****************************************************************************/
/*                            Programme principal                          */
/*                  Impression d'une matrice NB_LIGNES * NB_LIGNES         */
/****************************************************************************/
void main()
{
  affiche_matrice(iMat);
}
```

EXERCICE 14

```
#include<stdio.h>
/* Déclaration d'un tableau contenant le nombre de jours de tous les mois  */
/* de l'année, l'élément 0 étant inutilisé                                 */
int iNb_jours[13];

/* Fonction qui Initialise le tableau iNb_jours            */
/* Paramètres d'entrée : aucun                             */
/* Paramètres de sortie : rien                             */
void initialise_iNb_jours()
{ /* Déclaration d'un compteur de mois */
  int iNumMois;

  /* Pour tous les mois de l'année */
  for(iNumMois=1;iNumMois<=12;iNumMois++)
  { /* Si le mois est février */
    if(iNumMois==2)
      /* Alors le nombre de jours est 28 */
      iNb_jours[iNumMois]=28;
    /* Sinon, si le numéro du mois est pair et <=7 ou impair et >7 */
    else if(((iNumMois % 2 == 0) && iNumMois<=7) || ((iNumMois % 2 != 0) && iNumMois>7))
      /* Alors le nombre de jours est 30 */
      iNb_jours[iNumMois]=30;
    /* Sinon le nombre de jours est de 31 */
    else iNb_jours[iNumMois]=31;
  }
}

/* Fonction qui affiche le tableau iNb_jours               */
/* Paramètres d'entrée : aucun                             */
/* Paramètres de sortie : rien                             */
void affiche_iNb_jours()
{ /* Déclaration d'un compteur de mois */
  int iNumMois;

  /* Pour tous les mois de l'année */
  for(iNumMois=1;iNumMois<=12;iNumMois++)
    /* Affichage de nombre de jours pour le mois en cours */
    printf("Le mois No. %d de l'année a %d jours\n",iNumMois,iNb_jours[iNumMois]);
}

/***************************************************************************/
/*                          Programme principal                            */
/***************************************************************************/
void main()
{ /* initialisation de iNb_jours */
  initialise_iNb_jours();
  /* Affichage de contenu du tableau */
  affiche_iNb_jours();
}
```

EXERCICE 15

```
#include<stdio.h>

/* Déclaration des variables globales heures, minutes et secondes */
int iHeures, iMinutes, iSecondes;

/* Fonction qui imprime l'heure                                    */
/* Utilisation des variables globales iHeures, iMinutes, iSecondes */
/* Paramètres d'entrée : aucun                                     */
/* Paramètres de sortie : rien                                     */
void affiche_heure()
{ /* Affichage de l'heure */
  printf("Il est %d heure", iHeures);

  /* Si il y a plus d'une heure, ajout d'un s à la fin de heure */
  if(iHeures>1) printf("s");

  /* Affichage des minutes*/
  printf(" %d minute", iMinutes);

  /* Si il y a plus d'une minute, ajout d'un s à la fin de minute */
  if(iMinutes>1) printf("s");

   /* Affichage des secondes */
  printf(" %d seconde", iSecondes);

  /* Si il y a plus d'une seconde, ajout d'un s à la fin de seconde */
  if(iSecondes>1) printf("s");

  printf("\n");
}

/* Fonction qui met l'heure à une certaine valeur                     */
/* Utilisation des variables globales iHeures, iMinutes, iSecondes    */
/* Paramètres d'entrée :  int iH : heures, valeur donnée à iHeures    */
/*                        int iM : minutes, valeur donnée à iMinutes  */
/*                        int iS : secondes, valeur donnée à iSecondes */
/* Paramètres de sortie : rien                                        */
void initialise_heure(int iH,int iM,int iS)
{ iHeures = iH; iMinutes=iM; iSecondes=iS;
}
```

```
/* Fonction qui incrémente l'heure d'une seconde                          */
/* Utilisation des variables globales iHeures, iMinutes, iSecondes       */
/* Paramètres d'entrée :  aucun                                           */
/* Paramètres de sortie : rien                                            */
void tick()
{ /* On ajoute une seconde à l'heure */
  iSecondes+=1;

  /* S'il y a plus de 60 secondes   */
  if(iSecondes>=60)
  { /* On remet le nombre de secondes à zéro */
    iSecondes = 0;
    /* On incrémente le nombre de minutes */
    iMinutes+=1;

    /* S'il y a plus de 60 minutes  */
    if(iMinutes>=60)
    { /* On remet le nombre de minutes à zéro */
      iMinutes = 0;
      /* On incrémente l'heure */
      iHeures+=1;

      /* S'il y a plus de 24 heures, on remet l'heure à zéro */
      if(iHeures>=24) iHeures = 0;


    } /* Fin du if(iMinutes>60)*/

  } /* Fin du if(iSecondes>60)*/
}

/****************************************************************************/
/*                             Programme principal                          */
/****************************************************************************/
void main()
{ /* jeu d'essai 1 */
  initialise_heure(3,32,10);
  /* Incrémentation de l'heure d'une seconde */
  tick();
  /* Affichage */
  affiche_heure();

  /* jeu d'essai 2 */
  initialise_heure(1,32,59);
  /* Incrémentation de l'heure d'une seconde */
  tick();
  /* Affichage */
  affiche_heure();

  /* jeu d'essai 4 */
```

```
  initialise_heure(3,59,59);
  /* Incrémentation de l'heure d'une seconde */
  tick();
  /* Affichage */
  affiche_heure();

  /* jeu d'essai 5 */
  initialise_heure(23,59,59);
  /* Incrémentation de l'heure d'une seconde */
  tick();
  /* Affichage */
  affiche_heure();
}
```

EXERCICE 16

```
#include<stdio.h>
/* Fonction qui saisi un tableau d'entiers                                  */
/* Paramètres d'entrée :  int iTab[]: tableau d'entiers à initialiser  */
/*                        int iNb : Nombre d'éléments du tableau        */
/* Paramètres de sortie : rien                                          */
void saisir(int iTab[],int iNb)
{ /* Déclaration d'un compteur d'éléments du tableau */
  int iElement;

  /* Pour tous les éléments du tableau */
  for(iElement=0;iElement<iNb;iElement++)
  { /* Affichage d'un message de saisi pour l'utilisateur */
    printf("iTab[%d]=",iElement);
    /* Saisi de l'utilisateur */
    scanf("%d",&iTab[iElement]);
  }
}

/* Fonction qui affiche un tableau d'entiers                               */
/* Paramètres d'entrée :  int iTab[]: tableau d'entiers à afficher     */
/*                        int iNb : Nombre d'éléments du tableau        */
/* Paramètres de sortie : rien                                          */
void afficher(int iTab[],int iNb)
{ /* Déclaration d'un compteur d'éléments du tableau */
  int iElement;

  /* Pour tous les éléments du tableau */
  for(iElement=0;iElement<iNb;iElement++)
  { /* Affichage de l'élément */
    printf("%d ",iTab[iElement]);
  }
}
```

```
/* Fonction qui trie un tableau d'entiers                                 */
/* Paramètres d'entrée :  int iTab[]: tableau d'entiers à afficher        */
/*                        int iNb : Nombre d'éléments du tableau          */
/* Paramètres de sortie : rien                                            */
void trier(int iTab[],int iNb)
{ /* Déclaration de deux compteurs :                                    */
  /* iCompteur1 compte les éléments du tableau du 0ème au iNbème -1     */
  /* iCompteur2 compte les éléments du tableau de iCompteur+1 à iNb     */
  int iCompteur1,iCompteur2;

  /* Déclaration de la valeur d'un élément du tableau */
  int iVal;

  /* Pour tous les éléments du tableau du 0ème à l'avant-dernier */
  for(iCompteur1=0;iCompteur1<iNb-1;iCompteur1++)

     /* Pour tous les éléments du tableau qui suivent l'index iCompteur1 */
     for(iCompteur2=iCompteur1+1;iCompteur2<iNb;iCompteur2++)

       /* Si l'élément positionné à l'index iCompteur1 est plus grand   */
       /* que celui positionné à l'index iCompteur2                     */
       if(iTab[iCompteur1]>iTab[iCompteur2])
       { /* Echange des élément */
         iVal=iTab[iCompteur1];
         iTab[iCompteur1]=iTab[iCompteur2];
         iTab[iCompteur2]=iVal;
       }
}

/****************************************************************************/
/*                           Programme principal                            */
/****************************************************************************/
void main()
{ /* Déclaration d'un tableau de 20 entiers */
  int iTab[20];

  /* Initialisation du tableau */
  saisir(iTab,20);
  /* Tri du tableau */
  trier(iTab,20);
  /* Affichage du tableau trié */
  afficher(iTab,20);
}
```

EXERCICE 20

```
#include<stdio.h>
/* Déclaration d'une constante contenant le caractère de fin de chaîne */
const char NULL_C = '\0';
/* Déclaration et initialisation de deux chaînes de caractères globales */
char cTab1[] = "Cette chaîne comporte 35 caractères";
char cTab2[] = "et celle ci fait 30 caractères";

/* Fonction qui calcule la longueur d'une chaîne de caractères              */
/* Paramètres d'entrée :  char cChaine[] : chaîne dont on calcule la longueur */
/* Paramètres de sortie : nombre de caractères (int)                        */
int longueur_chaine1(char cChaine[])
{ /* Déclaration et initialisation d'un Compteur */
  int iLongueur =0;

  /* Tant que l'on arrive pas à la fin de la chaîne, on incrémente le compteur */
  while(cChaine[iLongueur] != NULL_C) iLongueur++;
  /* équivalent à while(cChaine[iLongueur]) iLongueur++; */

  /* On retourne la longueur de la chaîne */
  return(iLongueur);
}

/* Fonction qui calcule la longueur d'une chaîne de caractères               */
/* Paramètres d'entrée :                                                     */
/*               char *cPointeur_chaine : chaîne dont on calcule la longueur  */
/* Paramètres de sortie : nombre de caractères (int)                         */
int longueur_chaine2(char *cPointeur_chaine)
{ /* Déclaration et initialisation d'un Compteur */
  int iLongueur =0;

  /* Tant que le caractère pointé n'est pas la caractère de fin de chaîne  */
  while(*cPointeur_chaine != NULL_C)
  { /* On incrémente le compteur */
    iLongueur ++;
    /* On passe au caractère suivant */
    cPointeur_chaine++;
  }

  /* On retourne la longueur de la chaîne */
  return(iLongueur);
}

/*************************************************************************/
/*                           Programme principal                         */
/*************************************************************************/
void main()
{ printf("La longueur de iTab1 est %d\n",longueur_chaine1(cTab1));
  printf("La longueur de iTab2 est %d",longueur_chaine2(cTab2));
}
```

EXERCICE 21

```
#include<stdio.h>

/* Déclaration et initialisation d'un message global au programme */
char cMessage [] = "Les sanglots longs des violons de l'automne\n
blessent mon coeur d'une langueur monotone";

/* Déclaration d'une constante de décalage */
const int DECALAGE = 5;
/* Déclaration de constantes permettant de savoir si on est dans */
/* l'alphabet des minuscules ou des majuscules                   */
const int MINUS = 0
const  int MAJ = 1

/* Fonction qui crypte un message                                           */
/* Paramètres d'entrée :                                                    */
/*              char *cPointeur_chaine : chaîne à crypter                   */
/* Paramètres de sortie : rien                                              */
void crypt(char *cPointeur_chaine)
{ /* Définition d'un indicateur du type des lettres (minuscules ou majuscules */
  int iIndicateur;

  /* Si la lettre pointée est une minuscule */
  if(*cPointeur_chaine>='a' && *cPointeur_chaine<='z')
    /* On initialise l'indicateur à minuscules */
    iIndicateur=MINUS;

  /* Sinon si la lettre pointée est une majuscule */
  else if(*cPointeur_chaine>='A' && *cPointeur_chaine<='Z')
    /* On initialise l'indicateur à majuscules */
    iIndicateur=MAJ;

  /* Sinon, la lettre pointée n'est ni une majuscule, ni une minuscule */
  /* On sort de la procédure                                           */
  else return;

  /* On modifie la valeur de la lettre pointée par la valeur de la lettre situé */
  /* à DECALAGE lettre plus loin                                                */
  *cPointeur_chaine = *cPointeur_chaine + DECALAGE;

  /* Si la lettre pointée est minuscule et après 'z' ou majuscule et après 'Z'*/
   if((iIndicateur==MINUS && *cPointeur_chaine>'z')
       || (iIndicateur==MAJ && *cPointeur_chaine>'Z'))
    /* On décrémente la valeur de la lettre de 26 */
    *cPointeur_chaine = *cPointeur_chaine -26;
}
```

```
/****************************************************************************/
/*                           Programme principal                            */
/****************************************************************************/
void main()
{ /* Déclaration d'un pointeur sur un entier */
  char* cPointeur = NULL;
  /* Déclaration d'un compteur */
  int iCompteur;

  /* Phase de cryptage */
  /* Initialisation du pointeur sur la première lettre du message */
  cPointeur=&cMessage[0];

  /* Tant que le pointeur pointe sur un caractère */
  while(*cPointeur)
    /* Cryptage de la lettre */
    crypt(cPointeur++);

  /* Impression du résultat */
  printf("résultat :\n");
  /* Initialisation du compteur à zéro */
  iCompteur=0;
  /* Tant qu'il y a un caractère dans le message, on l'affiche et */
  /* on incrémente le compteur                                     */
  while(cMessage[iCompteur]) printf("%c",cMessage[iCompteur++]);

}
```

EXERCICE 22

```
#include<stdio.h>

/*************************************************************************/
/*                           Programme principal                         */
/*Compte les chiffres, caractères d'espacement et tout autre carac en entrée */
/*************************************************************************/
void main()
{ /* Déclaration d'un tableau permettant d'enregistrer le nombre */
  /* de chiffres de 1 à 9 rencontrés en entrée                   */
  int iNbChiffres[10];
  /* Déclaration d'un compteur d'espace, d'un compteur d'autres caractères */
  int iNbEspaces, iNbAutres;
  /* Déclaration d'un caractère */
  char iCaractere;
  /* Déclaratio d'un compteur */
  int iCompteur;

  /* Initialisation des compteurs d'espaces et autres */
  iNbEspaces=iNbAutres=0;

  /* Initialisation du tableau permettant de compter les chiffres */
  for(iCompteur=0;iCompteur<10;iCompteur++)
   iNbChiffres[iCompteur]=0;

  /* Tant qu'un caractère est saisi */
  while((iCaractere=getchar()) != EOF)
      /* S'il s'agit d'un espace ou d'un retour chariot ou d'une tabulation */
      if(iCaractere==' ' || iCaractere=='\n' || iCaractere=='\t')
         /* Incrémentation du nombre d'espaces */
         iNbEspaces++;

      /* S'il ne s'agit pas d'un espace mais d'un chiffre */
      else if(iCaractere>='0' && iCaractere<='9')
         /* Incrémentation de la case correspondant au chiffre rencontré */
         iNbChiffres[iCaractere-'0']++;

      /* Si le caractère n'est ni un espace ni un chiffre */
      else
         /* Incrémentation des autres caractères */
         iNbAutres++;

     /* Affichage du tableau comptant le nombre de chiffres rencontrés */
     printf("\nChiffres : ");

     for(iCompteur=0;iCompteur<10;iCompteur++)
        printf("%d ",iNbChiffres[iCompteur]);
     printf("Espacements : %d Autres : %d\n",iNbEspaces, iNbAutres);
}
```

EXERCICE 23

```
#include<stdio.h>

/* Fonction de lecture d'une chaîne                                           */
/* Paramètres d'entrée :  char* cChaine : chaîne à lire                       */
/* Paramètres de sortie : rien                                                */
void lecture(char* cChaine)
{ /* Saisi d'une chaîne sans blanc au clavier */
  printf("nom :"); scanf("%s",cChaine);
}

/* Fonction de comparaison de chaînes                                         */
/* Paramètres d'entrée : char* cChaine : chaîne à lire                        */
/* Paramètres de sortie : 1 (si la suite n'est pas finie (mot fin entré) ou 0 */
int comparaison(char* cChaine)
{
  /* Déclaration d'une variable statique                                    */
  /* Une variable statique a une durée de vie égale à celle du programme */
  /* A chaque nouveau passage dans la procédure comparaison, istaticTotal */
  /* reprendra la valeur qu'elle avait lors du dernier passage dans la    */
  /* fonction. Cette variable n'est pas utilisable dans le main.         */
  static int istaticTotal;

  /* Si la chaîne est le mot fin */
  if((strcmp(cChaine,"fin"))==0)
  { /* Affichage du total */
    printf("Total de noms de plus de dix caractères = %d\n",istaticTotal);
    /* Sortie de la fonction */
    return 0;
  }

  /* Si la chaîne contient plus de dix caractères */
  if(strlen(cChaine)>10)
    /* Incrémentation de total */
    istaticTotal ++;
  /* Sortie de la fonction */
  return 1;
}

/****************************************************************************/
/*                          Programme principal                             */
/****************************************************************************/
void main()
{ /* Déclaration d'un tableau de caractères */
  char cChaine[20];
  /* Boucler */
  do { lecture(cChaine); /* Lire la chaîne */
     } /* Tant que comparaison retourne 1 */
  while(comparaison(cChaine))
}
```

EXERCICE 24

```
#include<stdio.h>
#include <stdlib.h>
/* Fonction de menu                                                          */
/* Paramètres d'entrée : aucun                                               */
/* Paramètres de sortie : char le caractère saisi par l'utilisateur          */
/*                        (son choix dans le menu)                           */
char menu()
{
  /* Affichage du menu */
  printf("\n1-----> Saisie\n");
  printf("2-----> Affichage\n");
  printf("3-----> Nombre de mots\n");
  printf("4-----> Inversion\n");
  printf("0-----> Sortie\n");
  /* On retourne et on affiche le choix de l'utilisateur */
  return(getchar());
}

/* Fonction de saisie d'une chaîne                                           */
/* Paramètres d'entrée : aucun                                               */
/* Paramètres de sortie : char* : la chaîne saisie                           */
char* saisir()
{ /* Déclaration d'un tableau de caractères et d'une chaîne */
  char cTableau[80];
  char* cChaine = NULL;


  /* Saisie de la chaîne */
  printf("\nchaîne :");
  gets(cTableau);
  /* Allocation mémoire de la chaîne cChaine à la taille du tableau */
  cChaine = (char*)malloc(strlen(cTableau) + 1);
  /* Copie du contenu du tableau dans la chaîne */
  strcpy(cChaine,cTableau);
  /* On retourne la chaîne */
  return cChaine;
}

/* Fonction d'affichage d'une chaîne                                         */
/* Paramètres d'entrée : char cTableau[] : chaîne à afficher                 */
/* Paramètres de sortie : rien                                               */
void afficher(char cTableau[])
{ /* Affichage */
  puts("\n");
  puts(cTableau);
}
```

```
/* Fonction qui inverse une chaîne                                        */
/* Paramètres d'entrée : char* cChaine: chaîne à inverser                 */
/* Paramètres de sortie : rien                                            */
void inverser(char* cChaine)
{ /* Déclaration de deux compteurs */
  int iCompteur1,iCompteur2;
  /* Déclaration d'un caractère tampon */
  char cCaractere;

  /* Pour iCompteur1 allant du début à la fin de la première moitié     */
  /* de la chaîne :                                                     */
  /* On va échanger les caractères du début avec ceux de la fin en      */
  /* partant de chaque bout de la chaîne et en remontant vers le milieu */
  /* iCompteur2 = longueur de la chaîne pour éviter de la calculer      */
  /* à chaque passage de boucle (un seul appel à la fonction strlen)    */
 for(iCompteur1=0,iCompteur2=strlen(cChaine);iCompteur1<iCompteur2/2;iCompteur1++)
     { /* On enregistre le caractère situé à l'index iCompteur1 */
       cCaractere = cChaine[iCompteur1];
       /* On modifie la valeur de ce caractère dans la chaîne par celui situé */
       /* à l'autre extrémité de la chaîne mais au même niveau que iCompteur1 */
       cChaine[iCompteur1] = cChaine[iCompteur2-iCompteur1-1];
       /* Et on remplace le caractère situé à la fin par celui qui était à     */
       /* iCompteur1 et enregistré dans cCaractere                             */
       cChaine[iCompteur2-iCompteur1-1] = cCaractere;
     }
}

/* Fonction qui compte les mots d'une chaîne                              */
/* Paramètres d'entrée : char* cTableau : chaîne à traiter                */
/* Paramètres de sortie : int : nombre de mots de la chaîne               */
int mots(char *cTableau)
{ /* Déclaration de deux compteurs */
  int iCompteur1,iCompteur2;
  /* Déclaration et initialisation d'un compteur de mots */
  int iNbMots = 0;

/* Pour iCompteur1 allant du début à la fin de la chaîne */
for(iCompteur1=0,iCompteur2=strlen(cTableau);iCompteur1<iCompteur2;iCompteur1++)
 /* Si on a un espace suivi d'un espace ou du caractère de fin de chaîne*/
 if((cTableau[iCompteur1+1]==' '||cTableau[iCompteur1 + 1]=='\0')&&cTableau[iCompteur1]!=' ')
    /* Alors on incrémente le nombre de mots */
    iNbMots++;

  /* On retourne le nombre de mots */
  return iNbMots;
}
```

```
/****************************************************************************/
/*                              Programme principal                         */
/****************************************************************************/
void main()
{ /* Déclaration d'un caractère pour enregistrer le choix de l'utilisateur */
  char cChoix;
  /* Déclaration et initialisation d'une chaîne */
  char* cChaine="";

  /* Tant que le choix n'est pas Sortie */
  while((cChoix=menu()) !='0')
  { /* Saisi du retour chariot tape par l'utilisateur après le menu */
    getchar();

    /* On teste le choix de l'utilisateur */
    switch(cChoix)
    {
      case '1':cChaine=saisir();break;

      case '2':afficher(cChaine);break;

      case '3':printf("\nNombre de mots = %d\n",mots(cChaine));break;

      case '4':inverser(cChaine);break;

      default: printf("Choix erroné\n");

    } /* Fin du switch*/

    printf("\nFrappez une touche pour revenir au menu "); getchar();
  } /* Fin du while */
}
```

EXERCICE 25

```
#include<stdio.h>
#include<stdlib.h>

/* Fonction de lecture d'un tableau de chaînes de caractères                        */
/* Paramètres d'entrée : char *cTableau[] : tableau de chaînes de caractères        */
/*                       int iNbMax : nombre de chaînes maximum du tableau          */
/* Paramètres de sortie : int : nombre de chaînes lues et insérées dans le tableau */
int lecture(char* cTableau[],int iNbMax)
{ /* Déclaration de trois compteurs : un pour les chaînes lues, */
  /* deux pour les index du tableau                             */
  int iNbChaines=0,iIndex,iPositionChaine;

  /* Déclaration d'une chaîne à saisir */
  char cChaine[80];

  /* Tant qu'il y a moins de iNbMax chaînes lues et que le mot fin n'a pas été saisi */
  while(iNbChaines<iNbMax && strcmp(gets(cChaine),"fin"))
  {
    /* On recherche la position de la chaîne saisie dans le tableau en comparant */
    /* la chaîne saisie et celles contenues dans le tableau                      */
    for(iPositionChaine=0;(iPositionChaine<iNbChaines)
       && (strcmp(cTableau[iPositionChaine], cChaine)<0)); iPositnChaine++)
      /* On ne fait rien dans le corps de la boucle */
      ;

    /* On décale les chaînes présentes dans le tableau (en partant de la fin du tableau    */
    /* pour y insérer la nouvelle chaîne à la position iPositionChaine, trouvée précédemment*/
    for(iIndex=iNbChaines;iIndex>iPositionChaine;iIndex--)
       /* On déplace l'élément courant d'un cran */
       cTableau[iIndex] = cTableau[iIndex-1];

    /* Allocation mémoire de la place pour insérer la nouvelle chaîne dans le tableau */
    cTableau[iIndex] = (char*)malloc(strlen(cChaine)+1);
    /* copie de la chaîne dans le tableau */
    strcpy(cTableau[iIndex],cChaine);

    /* Incrémentation du compteur de chaînes */
    iNbChaines++;

  } /* Fin du while */

  /* On retourne le nombre de chaînes saisies */
  return iNbChaines;
}
```

```
/* Fonction d'affichage d'un nombre fixe de chaînes contenues dans un tableau     */
/* Paramètres d'entrée : char* cTableau[] : tableau de chaînes de caractères      */
/*                       int iNbMax : nombre de chaînes maximum à afficher        */
/* Paramètres de sortie : rien                                                    */
void ecriture(char *cTableau[],int iNbMax)
{ /* Déclaration d'un Compteur */
  int iCompteur;

  /* Pour tous les éléments du tableau */
  for(iCompteur=0;iCompteur<iNbMax;iCompteur++)
    /* Affichage de la chaîne */
    puts(cTableau[iCompteur]);
}

/*********************************************************************************/
/*                             Programme principal                                 */
/*********************************************************************************/
void main()
{ /* Déclaration d'un tableau de 20 chaînes, et d'un compteur */
  char* cTableau[20];
  int iNbChainesLues;

  /* Lecture des chaînes */
  iNbChainesLues=lecture(cTableau,20);
  /* Affichage */
  printf("\n");
  ecriture(cTableau,iNbChainesLues);
}
```

Pour la question 2, il suffit d'insérer l'instruction $ecriture(cTableau, iNbChaines)$ dans la fonction lecture, avant de retourner à l'itération.

EXERCICE 29

```
#include "stdio.h"
/***************************************************************************/
/*                              programme principal                        */
/*            Lecture et impression du contenu d'un fichier                */
/*                      d'enregistrements de commandes                     */
/***************************************************************************/
void main()
{
  /* Déclaration du fichier de données*/
  FILE* FCommandes;

  /* Déclaration des champs de la commande */
  char cNom[80];
  char cArticle[80];
  int iNombre,iPrix;

   /* Ouverture du fichier */
   FCommandes = fopen("Td2_exo11.data","r");

   /* Si le fichier ne peut pas s'ouvrir en lecture */
   if (FCommandes == NULL)
      /* Affichage d'un message d'erreur */
      printf("Impossible d'ouvrir le fichier Td2_exo11.data\n");

   /* Si le fichier peut s'ouvrir en lecture */
   else
   {
     /* Tant qu'on est pas à la fin du fichier, on lit les enregistrements */
     while(fscanf(FCommandes,"%s %s %d %d",cNom,cArticle,&iNombre,&iPrix) != EOF)
       /* Affichage du contenu du fichier */
      printf("%s %s %d\n",cNom,cArticle,iNombre * iPrix);

      /* Fermeture du fichier*/
      fclose(FCommandes);

   } /* Fin du else*/
}
```

EXERCICE 30

```
#include <stdio.h>

/**************************************************************************/
/*                           programme principal                          */
/*             Lecture et impression du contenu d'un fichier              */
/*                    d'enregistrements de commandes                      */
/*                        a l'aide d'une structure                        */
/**************************************************************************/
void main()
{ /* Déclaration du fichier de données*/
  FILE* FCommandes;

  /* Définition d'une structure de commande */
  struct commande
   { char cNom[80];
     char cArticle[80];
     int iNombre,iPrix;
   };

  /* Déclaration d'une constante nombre de commande max */
  const int iNB_COM = 100;

  /* Déclaration d'un tableau de commandes */
  struct commande tab_com[iNB_COM];

  /* Déclaration d'un index pour le tableau */
  int iIndex;

  /* Déclaration d'un index pour le dernier élément valide dans le tableau */
  /* après remplissage                                                     */
  int iDerniere;

  /* Si le fichier ne peut pas s'ouvrir en lecture */
  if ((FCommandes = fopen("Td2_exo11.data","r")) == NULL)
      /* Affichage d'un message d'erreur */
      printf("Impossible d'ouvrir le fichier Td2_exo11.data\n");
  /* Si le fichier peut s'ouvrir en lecture */
  else
  {
   /* Boucle de lecture des commandes   */

   /* Initialisation de l'index */
   iIndex = 0;
```

```
   /* tant que l'index est inférieur au nombre max de commandes et   */
   /*  qu'on est pas à la fin du fichier, on lit les enregistrements */
   while(iIndex < iNB_COM && fscanf(FCommandes,"%s %s %d %d",
                             tab_com[iIndex].cNom,
                             tab_com[iIndex].cArticle,
                             &tab_com[iIndex].iNombre,
                             &tab_com[iIndex].iPrix) != EOF)

      /* Incrémentation de l'index */
      iIndex++;

   /* S'il y a plus de 100 commandes */
   if (iIndex >= iNB_COM)
      printf("le tableau tab_com est sous-dimentionné\n");

   /* S'il y a moins de 100 commandes */
   else
      {
      /* Impression des commandes mémorisées   */

      /* Initialisation de l'index de la dernière commande */
      iDerniere = iIndex - 1;

      /* Pour toutes les Commandes */
      for (iIndex = 0; iIndex <= iDerniere; iIndex++)
         /* Affichage des commandes */
         printf("%s %s %d %d\n", tab_com[iIndex].cNom, tab_com[iIndex].cArticle,
                                 tab_com[iIndex].iNombre, tab_com[iIndex].iPrix);

      /* Fermeture du fichier */
      fclose(FCommandes);
      }
   }
}
```

EXERCICE 31

```
#include <stdio.h>
#include <stdlib.h>

/**************************************************************************/
/*                             programme principal                        */
/*              Lecture et impression du contenu d'un fichier             */
/*                       d'enregistrements de commandes                   */
/*                           a l'aide d'une liste chaînée                 */
/**************************************************************************/
void main()
{
  /* Déclaration du fichier de données*/
  FILE* FCommandes;

  /* Définition d'une structure de commande */
  struct commande
   { char cNom[80];
     char cArticle[80];
     int iNombre,iPrix;
     /* Pointeur sur l'élément suivant de la liste chaînée */
     struct commande *suiv;
   };

  /* Déclaration et initialisation de la liste de commandes */
  struct commande *l_com = NULL;
  /* Déclaration de pointeur sur la commande courante et la commande précédente */
  struct commande *prec,*cour;
  /* Déclaration de la valeur de retour du scanf */
  int iVal_ret;

  /* Ouverture du fichier */
   FCommandes = fopen("Td2_exo11.data","r");
  /* Si le fichier ne peut pas s'ouvrir en lecture */
  if (FCommandes == NULL)
      /* Affichage d'un message d'erreur */
      printf("Impossible d'ouvrir le fichier Td2_exo11.data\n");
  /* Si le fichier peut s'ouvrir en lecture */
  else
  {
   /* Création de la liste chaînée de commandes */

   /* Boucler tant qu'il y a des lignes dans le fichier */
   do
   {
     /* Allocation mémoire du pointeur sur la commande courante */
     cour = (struct commande*)malloc(sizeof(struct commande));
```

```
    /* Lecture de la ligne de commande */
    iVal_ret = fscanf(FCommandes,"%s %s %d %d",
                            cour -> cNom,
                            cour -> cArticle,
                            &(cour -> iNombre),
                            &(cour -> iPrix));

    /* Si on est à la fin du fichier */
    if (iVal_ret == EOF)
    { /* Libération de la mémoire occupée par le pointeur sur la commande */
      /* courante                                                         */
      free(cour);

      /* Si la liste est non vide, alors le suivant de l'élément précédent */
      /* est nul                                                           */
      if(l_com != NULL) prec -> suiv = NULL;
     } /* Fin du if (iVal_ret == EOF)

     /* Si on est pas à la fin du fichier */
     else
     {  /* Si la liste est vide, alors la liste contient la commande courante */
        if (l_com == NULL) l_com = cour;
        /* Si la liste est non vide, le suivant de l'élément précédent est */
        /* l'élément courant                                               */
        else prec -> suiv = cour;
        /* L'élément précédent devient l'élément courant */
        prec = cour;
      } /* Fin du else de if (iVal_ret == EOF) */
  }
  while (iVal_ret != EOF); /* Tant qu'on est pas à la fin du fichier */


  /* Parcours de la liste avec impression   */
  /* Si la liste est vide */
  if (l_com == NULL)
     printf("La liste de commandes est vide\n");
  /* Si la liste est non vide */
  else
  {
     /* Pour tous les éléments de la liste */
     for (cour = l_com; cour != NULL; cour = cour -> suiv)
       /* Affichage de l'élément courant */
       printf("%s %s %d %d\n",
               cour -> cNom,cour -> cArticle,cour -> iNombre,cour -> iPrix);
  }

  /* Fermeture du fichier   */
  fclose(FCommandes);
  }
}
```

EXERCICE 32

```
#include <stdio.h>
#include <stdlib.h>

/* Type commun a toutes les procédures    */
/* Définition d'une structure de commande */
  struct commande
   { char cNom[80];
     char cArticle[80];
     int iNombre,iPrix;
     /* Pointeur sur l'élément suivant de la liste chaînée */
     struct commande *suiv;
   };


/* Fonction qui imprime une structure commande                      */
/* Paramètres d'entrée : struct commande *com : une commande        */
/* Paramètres de sortie : rien                                      */
void print_com(struct commande *com)
{
printf("%s %s %d %d\n",
       com -> cNom,com -> cArticle,com -> iNombre,com -> iPrix);
}

/* Fonction qui recherche la commande pour laquelle le produit             */
/* nombre * prix est le maximum                                            */
/* Paramètres d'entrée : struct commande *l_com : la liste des commandes   */
/* Paramètres de sortie : struct commande *: un pointeur vers la structure */
/*                        commande recherchée ou NULL si l_com est vide    */
struct commande *max_com(struct commande * l_com)
{
  /* Déclaration d'un pointeur sur la commande pour laquelle le produit  */
  /* nombre * prix est le maximum maximum                                */
  struct commande *pmax;
  /* Déclaration d'un pointeur sur la commande courante */
  struct commande *cour;
  /* Déclaration des valeur maximum et courante */
  int iVmax,iVcour;

  /* SI la liste est vide */
  if (l_com == NULL)
   /* On sort de la fonction et on retourne NULL */
   return(NULL);
```

```
  /* Si la liste est non vide */
  else
  { /* Le pointeur sur la commande max est initialisé à la 1ère commande */
    pmax = l_com;
    /* Initialisation de la valeur max */
    iVmax = (pmax -> iNombre) * (pmax -> iPrix);

    /* Pour toutes les commandes */
    for (cour = l_com -> suiv; cour != NULL; cour = cour ->suiv)
    { /* Initialisation de la valeur courante */
      iVcour = (cour -> iNombre) * (cour -> iPrix);
      /* Si la valeur courante est supérieure à la valeur max */
      if (iVcour > iVmax)
      { /* Le max est modifié */
        iVmax = iVcour;
        pmax = cour;
      } /* Fin du if (iVcour > iVmax)*/
    }/* Fin du for */

    /* On retourne l'élément maximum */
    return(pmax);
  } /* Fin de  else*/
}

/**************************************************************************/
/*                            programme principal                         */
/**************************************************************************/
void main()
{
  /* Déclaration du fichier de données*/
  FILE* FCommandes;
  /* Déclaration et initialisation de la liste  de commandes */
  struct commande *l_com = NULL;
  /* Déclaration de pointeurs sur la commande courante et la commande précédente */
  struct commande *prec,*cour;
  /* Déclaration de la valeur de retour du scanf */
  int iVal_ret;

  /* Ouverture du fichier */
   FCommandes = fopen("Td2_exo11.data","r");
  /* Si le fichier ne peut pas s'ouvrir en lecture */
  if (FCommandes == NULL)
      /* Affichage d'un message d'erreur */
      printf("Impossible d'ouvrir le fichier Td2_exo11.data\n");
```

```
/* Si le fichier peut s'ouvrir en lecture */
else
{ /* Création de la liste chaînée de commandes */

  /* Boucler tant qu'il y a des lignes dans le fichier */
  do
   { /* Allocation mémoire du pointeur sur la commande courante */
    cour = (struct commande*)malloc(sizeof(struct commande));

    /* Lecture de la ligne de commande */
    iVal_ret = fscanf(FCommandes,"%s %s %d %d",
                           cour -> cNom,
                           cour -> cArticle,
                           &(cour -> iNombre),
                           &(cour -> iPrix));

     /* Si on est à la fin du fichier */
     if (iVal_ret == EOF)
     {
       /* Libération de la mémoire occupée par le pointeur sur la commande */
       /* courante                                                         */
       free(cour);

       /* Si la liste est non vide, alors le suivant de l'élément précédent */
       /* est nul                                                           */
       if(l_com != NULL) prec -> suiv = NULL;
      } /* Fin du if (iVal_ret == EOF)

      /* Si on est pas à la fin du fichier */
      else
      {
      /* Si la liste est vide, alors la liste contient la commande courante*/
       if (l_com == NULL) l_com = cour;
       /* Si la liste est non vide, le suivant de l'élément précédent est */
       /* l'élément courant                                               */
       else prec -> suiv = cour;
       /* L'élément précédent devient l'élément courant */
       prec = cour;
      } /* Fin du else de if (iVal_ret == EOF) */
  }
  /* Tant qu'on est pas à la fin du fichier */
  while (iVal_ret != EOF);

 /* Parcours de la liste avec impression   */

 /* Si la liste est vide */
 if (l_com == NULL)
    printf("La liste de commandes est vide\n");
```

```
   /* Si la liste est non vide */
   else
   {
      /* Pour tous les éléments de la liste */
      for (cour = l_com; cour != NULL; cour = cour -> suiv)
        /* Affichage de l'élément courant */
        print_com(cour);

      /* Recherche et impression de la commande maximum   */
      printf("La commande maximum est :\n");
      print_com(max_com(l_com));
      }

   /* Fermeture du fichier   */
   fclose(FCommandes);
   }
}
```

EXERCICE 33

```
#include<stdio.h>
#include<stdlib.h>

/* Déclaration d'un type élément de liste chaînée */
typedef struct lis
{
 /* Valeur de l'élément */
 int iValeur;
 /* Pointeur sur l'élément suivant */
 struct lis* suivant;
} *element;

/* Déclaration d'un type liste chaînée */
typedef struct t
{
 /* pointeurs sur le premier élément de la liste, sur le dernier */
 /* et sur l'élément courant                                      */
 element prem,der,cour;
} *liste;

/* Fonction de création d'une liste */
/* Paramètres d'entrée : Aucun      */
/* Paramètre de sortie : une liste  */
liste creer_liste()
{
  /* Déclaration d'une liste locale */
  liste Liste_Entier;

  /* Allocation mémoire de la liste */
  Liste_Entier = (liste)malloc(sizeof(struct t));
```

```
  /* Allocation mémoire des pointeurs prem, der, et cour de la liste    */
  /* au départ, la liste est vide, les pointeurs ne pointent sur rien.  */
  /* Ceci est une allocation multiple                                   */
  Liste_Entier->prem=Liste_Entier->der=Liste_Entier->cour = NULL;

  /* La liste est retournée */
  return Liste_Entier;
}

/* Fonction d'insertion après la position courante d'un élément         */
/* dans une liste                                                       */
/* Paramètres d'entrée :                                                */
/*                  liste Liste_Entier : liste où a lieu l'insertion    */
/*                  int iVal  : valeur de l'élément                     */
/* Paramètre de sortie : rien                                           */
void inserer_apres(liste Liste_Entier,int iVal)
{ /* Element à insérer */
  element Element_Liste = NULL;

  /* Allocation mémoire de l'élément */
  Element_Liste = (element)malloc(sizeof(struct lis));

  /* Initialisation de la valeur de e */
  Element_Liste->iValeur = iVal;

  /* S'il n'y a pas d'élément dans la liste */
  if (Liste_Entier->prem == NULL)
  { /* prem, der et cour pointent sur e */
    Liste_Entier->prem=Liste_Entier->der=Liste_Entier->cour =  Element_Liste;

    /* Il n'y a pas d'élément suivant */
    Element_Liste->suivant = NULL;

  } /* Fin de S'il n'y a pas d'élément dans la liste */

  /* Si la liste est non vide */
  else

    /* Si l'élément courant n'est pas le dernier */
    if(Liste_Entier->cour!=Liste_Entier->der)
    {
      /* Element_Liste pointe sur l'élément suivant de l'élément courant */
      Element_Liste->suivant = Liste_Entier->cour->suivant;

      /* Le suivant de l'élément courant est e */
      Liste_Entier->cour->suivant = Element_Liste;

      /* L'élément courant est Element_Liste */
      Liste_Entier->cour = Element_Liste;
    } /* Fin de Si l'élément courant n'est pas le dernier */
```

```
  /* Si l'élément courant est le dernier élément */
  else
  { /* Element_Liste n'a pas de suivant */
    Element_Liste->suivant = NULL;

    /* L'élément suivant de l'élément courant est Element_Liste */
    Liste_Entier->cour->suivant = Element_Liste;

    /* Element_Liste est l'élément courant et le dernier élément */
    Liste_Entier->cour = Liste_Entier->der = Element_Liste;
  } /* Fin de Si l'élément courant est le dernier élément */
}

/***************************************************************************/
/*                          Programme principal                            */
/***************************************************************************/
void main()
{ /* Déclaration d'une liste */
  liste l;
  /* Déclaration d'un compteur */
  int iCompteur;

  /* Création de la liste */
  l = creer_liste();

  /* Pour les entiers de 1 à 5 */
  for(iCompteur=1;iCompteur<=5;iCompteur++)
  { /* Ajout de l'entier dans la liste */
    inserer_apres(l,iCompteur);
    /* Affichage de l'élément courant */
    printf("Courant = %d\n",l->cour->iValeur);
  }

  /* Affichage de la liste */
  /* On positionne le pointeur courant sur le premier élément */
  l->cour = l->prem;
  printf("Liste dans sa totalité :");

  /* Tant qu'on est pas au dernier élément */
  while(l->cour!=l->der)
  { /* Affichage de la valeur de l'élément courant */
    printf("%d ",l->cour->iValeur);
    /* On passe au suivant */
    l->cour = l->cour->suivant;
  }

  /* Affichage du dernier élément */
  printf("%d ",l->cour->iValeur);
}
```

EXERCICE 42

Structure pile et opérations élémentaires en C

```
#include<stdio.h>

/* Définition d'une structure de pile */
typedef struct pi
{ /* valeur du sommet de la pile*/
  int iValeur;
  /* queue de la pile */
  struct pi *prec;
} * pile;

/* Fonction de création de pile         */
/* Paramètres d'entrée :  Aucun         */
/* Paramètres de sortie : La pile créée */
pile creer_pile()
{
 /* On retourne un pointeur nul */
 return NULL;
}

/* Fonction de test pour savoir si la pile est vide                        */
/* Paramètres d'entrée :  pile p : une pile                                */
/* Paramètres de sortie : un entier (0 si la pile est vide, >0 sinon)      */
int vide(pile p)
{
 /* On retourne la valeur du test p est NULL                  */
 /* Si p est NULL alors le test (p==NULL) sera différent de 0  */
 /* Si p est non NULL alors le test (p==NULL) sera égal à 0    */
 return (p==NULL);
}

/* Fonction qui retourne le sommet de la pile   */
/* Paramètres d'entrée :  pile p : une pile     */
/* Paramètres de sortie : le sommet de la pile  */
int sommet(pile p)
{ /* Si p est non vide */
  if(p)
   /* On retourne la valeur du sommet */
   return p->iValeur;
  /* Si p est vide on retourne la dernière valeur négative */
  /* possible pour un entier codé sur 2 octets             */
  else return -32768;
}
```

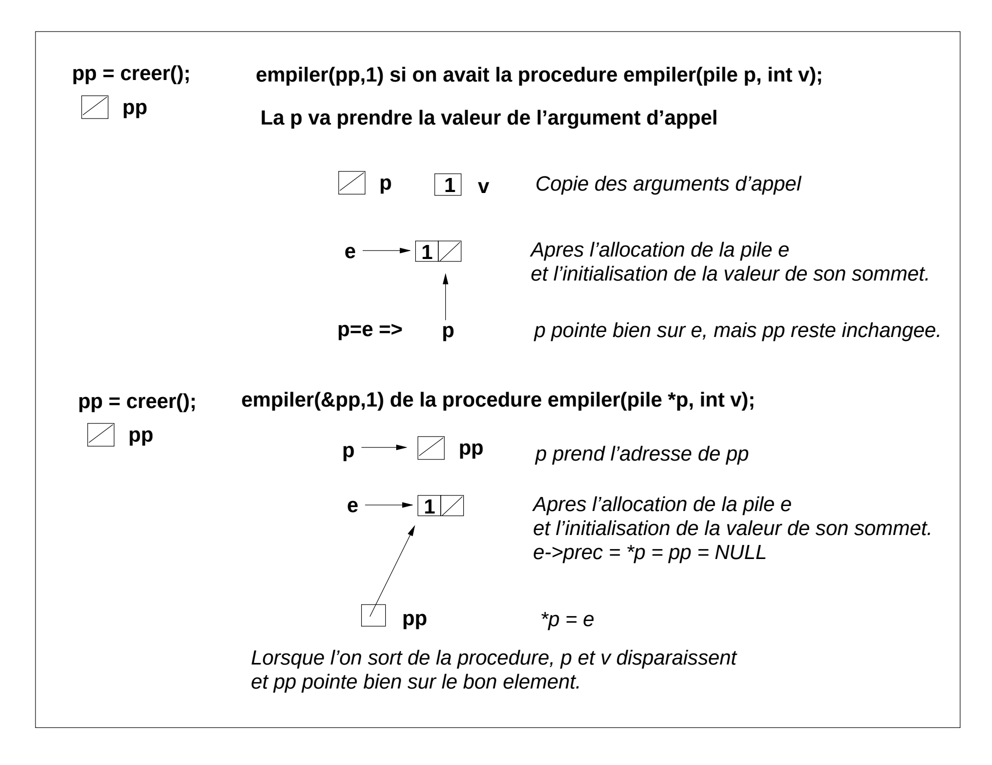


FIG. 2 – **Empilement d'une valeur dans une pile implémentée en liste chaînée.**

Fonction *empiler*

```
/* Fonction qui empile une valeur sur la pile              */
/* Paramètres d'entrée :                                   */
/*  pile *p  : un pointeur sur la pile qui va être modifiée */
/*  int iVal : la valeur à empiler                         */
/* Paramètres de sortie : rien                             */
void empiler(pile* p,int iVal)
{ /* Déclaration d'une pile locale à la fonction */
  pile e;

  /* Allocation mémoire de la pile locale */
  e = (pile)malloc(sizeof(struct pi));

  /* Initialisation de la valeur de la pile locale */
  e->iValeur = iVal;

  /* Ajout de la pile p à la queue de e */
  e->prec = *p;

  /* Mise à jour de la pile, p pointe sur le nouveau sommet de la pile */
  *p = e;
}
```

Pour les explications de la fonction *empiler*, voir figure 2.

Fonction désempiler

```
/* Fonction qui desempile la pile                              */
/* Paramètres d'entrée :                                       */
/*     pile* p : un pointeur sur la pile qui va être modifiée  */
/* Paramètres de sortie : rien                                 */
void desempiler(pile* p)
{ /* Déclaration d'une pile locale */
  pile q;

  /* Si la pile pointée par p est non vide */
  if(*p)
  { /* q est la pile pointée par p */
    q = *p;

    /* la pile pointée par p devient la queue de la pile */
    *p = (*p)->prec;

    /* Le mémoire occupée par q est libérée */
    free(q);
  }
}
```

Affichage récursif d'une pile L'affichage d'une pile peut se faire de manière récursive (voir figure 3). Une fonction récursive est une fonction qui s'appelle depuis son propre corps.
On remonte les appels en effectuant les actions écrites après les appels de récursivité.

```
/* Fonction qui affiche la pile de manière récursive      */
/* Paramètres d'entrée : pile p : la pile à afficher      */
/*   On ne transmet pas l'adresse de la pile, car elle ne */
/*   va pas être modifiée                                 */
/* Paramètres de sortie : rien                            */
void afficher_recursive(pile p)
{ /* Valeur du sommet de la pile à afficher */
  int iVal;

  /* Si la pile est non vide */
  if(p)
   {  /* iVal prend la valeur du sommet de la pile */
      iVal = p->iValeur;

      /* La pile devient la queue de p                            */
      /* la pile passée en paramètres n'est pas modifiée  pour autant */
      /* puisque ce n'est pas l'adresse qui est passée en paramètres  */
      p = p->prec;

      /* Appel récursif de la fonction pour le reste de la pile */
      afficher_recursive(p);
      /* Affichage de la valeur du sommet */
      printf("%d ", iVal);
   }
}
```

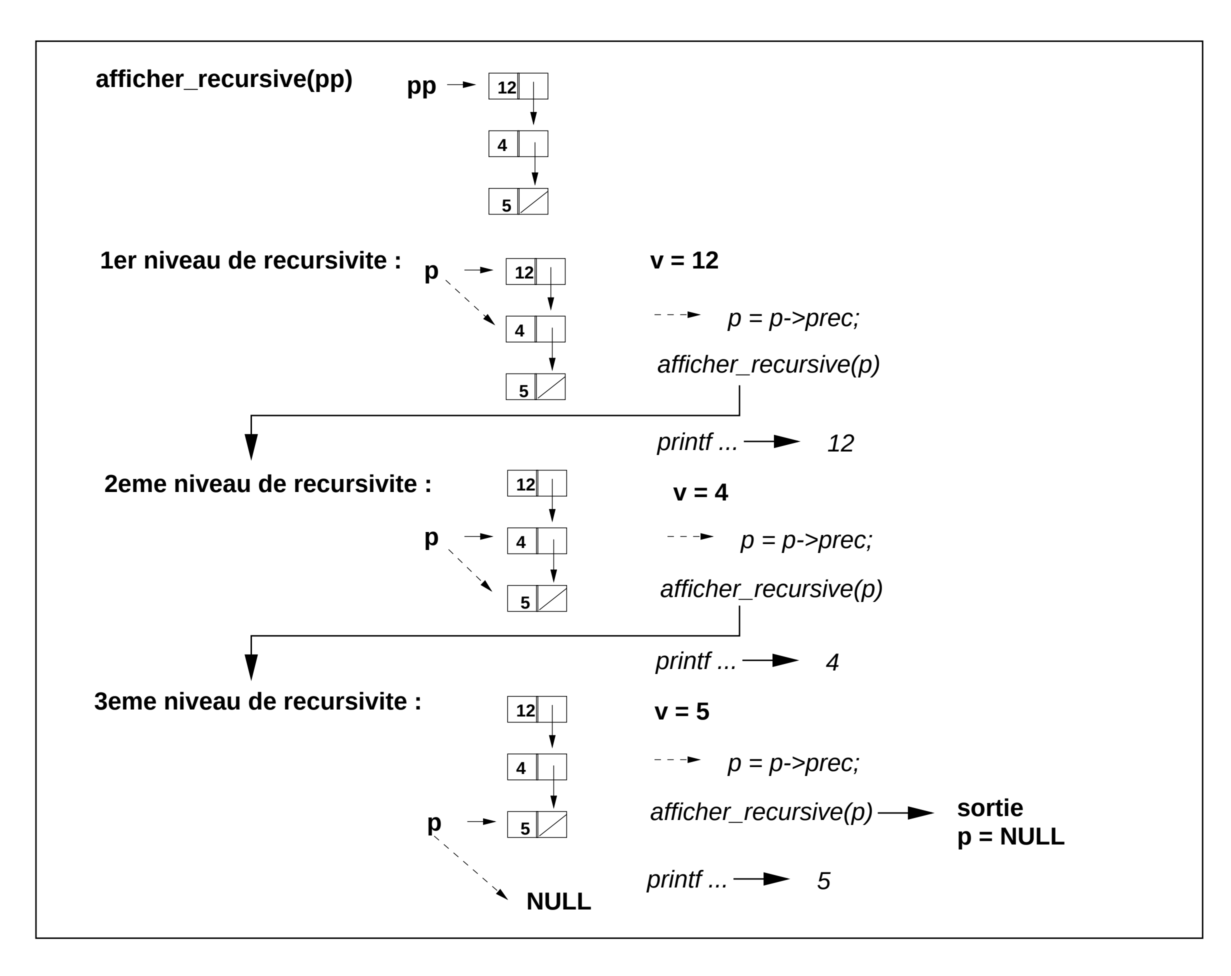


FIG. 3 – **Affichage récursif d'une pile implémentée en liste chaînée.**

Affichage d'une pile par échange de pointeurs

```
/* Fonction qui affiche la pile par échange de pointeurs  */
/* Paramètres d'entrée :                                  */
/*                     pile p : la pile à afficher        */
/*                     on ne transmet pas l'adresse de    */
/*                     la pile, car elle ne va pas être   */
/*                     modifiée                           */
/* Paramètres de sortie : rien                            */
void afficher_echange_pointeur(pile p)
{ /* Déclaration de deux piles locales */
  pile q,r;

  /* Initialisation de q */
  q = NULL;

  /* Tant que p est non nulle  */
  /* On va retourner la pile p */
  while(p)
   { /* Initialisation de r à l'élément précédente le sommet de p */
     r = p->prec;

     /* La queue de p devient q */
     p->prec = q;

     /* q devient la pile p modifiée précédemment */
     q = p;

     /* p devient la pile r initialisée précédemment*/
     p = r;
   }

  /* Tant que q est non nulle                */
  /* On va afficher les éléments de la pile q */
  while(q)
  { /* Affichage de la valeur du sommet de q */
    printf("%d ",q->iValeur);

    /* r devient la queue de la pile q */
    r = q->prec;

    /* La queue de q devient p */
    q->prec = p;

    /* la pile p devient q */
    p = q;

    /* la pile q devient la pile r */
    q = r;
    }
}
```

Lorsque l'on fait appel à la fonction *afficher_echange_pointeur*, l'adresse de la pile n'est pas modifiée. Seul le contenu de la pile est touché (voir figure 4 page 163).

```
/**************************************************************************/
/*                           Programme principal                          */
/**************************************************************************/
void main()
{ /* Déclaration d'une pile */
  pile p;

  /* Création de la pile */
  p=creer_pile();

  /* Empilement de 8, 4 et 19 */
  empiler(&p,8);
  afficher_recursive(p);
  printf("\n");
  empiler(&p,4);
  afficher_recursive(p);
  printf("\n");
  empiler(&p,19);
  afficher_recursive(p);
  printf("\n");


  /* desempilement */
  desempiler(&p);
  afficher_echange_pointeur(p);
  printf("\n");
  desempiler(&p);
  afficher_echange_pointeur(p);
  printf("\n");
  desempiler(&p);
  afficher_echange_pointeur(p);
  if(vide(p)) printf("Pile vide\n");

}
```

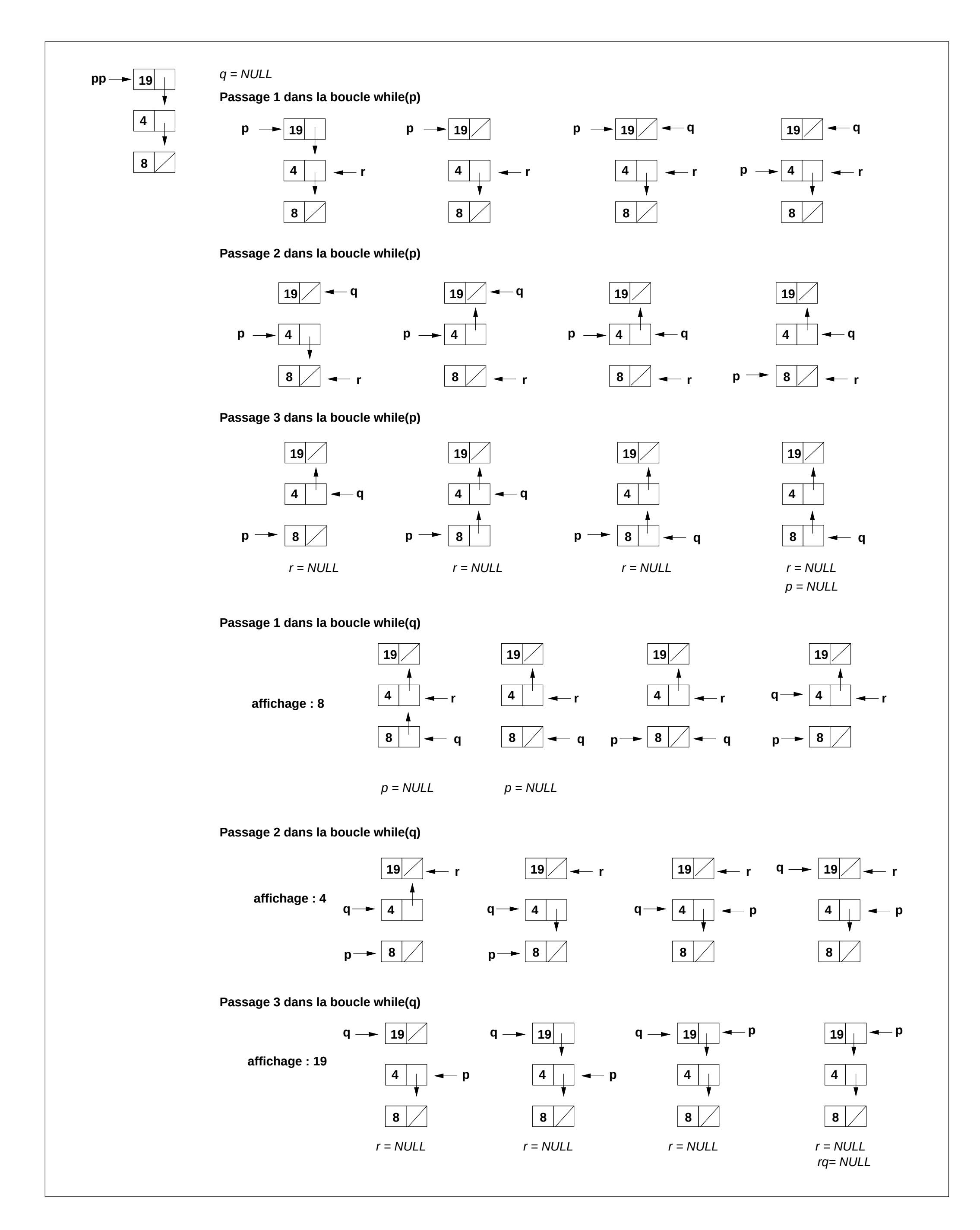


FIG. 4 – **Affichage par échange de pointeurs d'une pile implémentée en liste chaînée.**

EXERCICE 43

```
#include<stdio.h>

/* Définition d'un type file de caractères*/
typedef struct f
{ /* pointeurs sur le début et la fin de la zone mémoire de la file */
  char* cDebut;
  char* cFin;
  /* pointeurs sur la tête et la queue de la file                   */
  char* cTete;
  char* cQueue;
} *file;

/* Fonction de création de file                                           */
/* Paramètres d'entrée :                                                  */
/*          unsigned uTailleMemoire : longueur de la zone mémoire allouée */
/*                                    pour la file                        */
/* Paramètres de sortie : la file créée                                   */
file creer(unsigned uTailleMemoire)
{ /* Déclaration d'une file locale */
  file f;

  /* Allocation mémoire du pointeur sur la file */
  f = (file)malloc(sizeof(struct f));

  /* Allocation mémoire des pointeurs de début et de tête */
  f->cDebut = f->cTete = (char*)malloc(uTailleMemoire);

  /* Allocation mémoire des pointeurs de fin et de queue                 */
  /* On les placent à (n-1) espaces mémoire (de un octet) de la tête et    */
  /* du début                                                             */
  f->cFin = f->cQueue = f->cDebut + uTailleMemoire - 1;

  /* On retourne la file créée */
  return f;
}
```

```
/* Fonction de déplacement d'un espace mémoire d'un pointeur de la file   */
/* Paramètres d'entrée : file f : la file traitée                         */
/*                       char* cPointeur : le pointeur à avancer de 1     */
/* Paramètres de sortie : le pointeur avancé                              */
char* plusun(file f, char* cPointeur)
{ /* Si le pointeur est à la même adresse  */
  /* que la fin de zone mémoire de la file */
  if( cPointeur == f->cFin)
    /* On retourne le début de la zone mémoire */
    return f->cDebut;

  /* Si  le pointeur n'est pas à la même adresse que la fin de zone mémoire */
  else
     /* On avance le pointeur d'une case et on le retourne             */
     return ++cPointeur; /* Equivalent à cPointeur++; return cPointeur; */
}
/* Remarque si on appelle plusun(f,f->cQueue), f->cQueue n'est pas modifié  */
/* puisque ce n'est pas l'adresse de f->cQueue qui est passé en paramètre   */

/* Fonction de test permettant de savoir si la file est vide              */
/* Paramètres d'entrée : file f : la file traitée                         */
/* Paramètres de sortie : 0 si c'est faux, >0 sinon                       */
int vide(file f)
{ /* On retourne la valeur du test d'égalité entre l'emplacement qui suit */
  /* la queue de la file et l'adresse de la tête de la file               */
  return (plusun(f,f->cQueue) == f->cTete);
}

/* Fonction de test permettant de savoir si la file est pleine            */
/* Paramètres d'entrée : file f : la file traitée                         */
/* Paramètres de sortie : 0 si c'est faux, >0 sinon                       */
int pleine(file f)
{ /* On retourne la valeur du test d'égalité entre l'emplacement qui suit */
  /* la queue de la file  +1  et l'adresse de la tête de la file          */
  return (plusun(f,f->cQueue+1) == f->cTete);
}

/* Fonction de lecture de la tête de la file                              */
/* Paramètres d'entrée : file f : la file traitée                         */
/* Paramètres de sortie : la caractère de la tête de file                 */
char lire(file f)
{ /* Si la file est non vide                                               */
  /* NB : vide retourne 0 si la file est non vide, il faut donc tester si  */
  /* vide(f) est différent de 0                                            */
  if (!vide(f))
  { /* on retrouve le contenu de la tête de file */
    return *(f->cTete);
  }
}
```

```
/* Fonction d'affichage d'un message d'erreur           */
/* Paramètres d'entrée : char *cMess : message à afficher */
/* Paramètres de sortie : rien                          */
void erreur(char* cMess)
{ /* Affichage sur l'écran du message                      */
  /* puts est une fonction prédéfinie du C ~printf("%s".,.. */
  /* mais passe automatiquement a la ligne                 */
  puts(cMess);
}

/* Fonction d'avancement de la tête d'un espace mémoire   */
/* Paramètres d'entrée : file f : la file traitée         */
/* Paramètres de sortie : rien                            */
void avancer(file f)
{ /* Si la file est vide */
  if(vide(f))
    /* Affichage d'un message d'erreur */
    erreur("file vide!");
  /* Si la file est non vide */
  else
    /* on déplace la tête d'une case mémoire */
    f->cTete = plusun(f,f->cTete);
}

/* Fonction d'ajout d'un élément dans la file               */
/* Paramètres d'entrée : file f : la file traitée           */
/*                       char cCaractere : l'élément à ajouter */
/* Paramètres de sortie : rien                              */
void ajouter(file f, char cCaractere)
{ /* Si la file est pleine */
  if(pleine(f))
   /* Affichage d'un message d'erreur */
   erreur("File pleine!");

  /* Si la file n'est pas pleine */
  else
  { /* On déplace la queue de 1 */
    f->cQueue = plusun(f,f->cQueue);

    /* on ajoute la caractère à la queue de la file */
    *(f->cQueue) = cCaractere;
  }
}
```

```
/* Fonction d'affichage de la file                              */
/* Paramètres d'entrée : file f : la file traitée               */
/* Paramètres de sortie : rien                                  */
void afficher(file f)
{ /* Déclaration d'un pointeur sur un caractère */
  char* cPointeur;

  /* le pointeur local pointe sur la tête de la file */
  cPointeur = f->cTete;

  /* si la file est vide */
  if(vide(f))
   /* on affiche () */
   puts("()");

  /* si la file est non vide */
  else
  { /* On affiche ( */
    puts("(");

    /* Boucle infinie */
    while(1)
    { /* On affiche le caractère pointé par le pointeur local */
      putchar(*cPointeur);

      /* Si Pointeur pointe sur la queue de la file */
      if(cPointeur == f->cQueue)
         /* on sort de la boucle */
         continue;
      /* Si Pointeur ne pointe sur la queue de la file,             */
      /* on déplace Pointeur de 1 (pas besoin de else car on sort de  */
      /*  la boucle si le test de if est vrai)                      */
      cPointeur  = plusun(f,cPointeur);
    } /* fin du while */

    /* On affiche ) */
    puts ("\n)");
  } /* fin du si la file est non vide (else) */
}
```

```
/**************************************************************************/
/*                            Programme principal                         */
/**************************************************************************/
void main()
{ /* Déclaration d'une file */
  file f;

  /* Création d'une file */
  f=creer(4);

  /* Lecture de la file */
  printf("%c\n",lire(f));

  /* Avancement dans la file */
  avancer(f);

  /* Ajout du caractère A */
  ajouter(f,'A');
  afficher(f);
  printf("\n");
  /* Ajout du caractère B */
  ajouter(f,'B');
  afficher(f);
   printf("\n");
  /* Ajout du caractère C */
  ajouter(f,'C');
  afficher(f);
  printf("\n");
  printf("%c\n",lire(f));

}
```